FINAL REPORT

Robust Nonlinear Feedback Control of Aircraft Propulsion Systems

NASA Glen Research Center Grant: NAG-3-1975

SUBMITTED TO:

NASA Glen Research Center
Attention: Grants Office
21000 Brookpark Road, Mail Stop 500-319
Cleveland, OH 44135-3191

NASA TECHNICAL OFFICER:

Jonathan Litt
NASA Glen Research Center
21000 Brookpark Road, Mail Stop 77-1
Cleveland, OH 44135-3191
216.433.3748

15 JANUARY, 2001

Principal Investigators: Drs. William L. Garrard and Gary J. Balas
Department of Aerospace Engineering and Mechanics
University of Minnesota
Minneapolis, MN 55455
612.625.8000
wgarrard,balas@aem.umn.edu

Contract Dates: 1 November 1996 – 30 January 2001

Total Budget: $ 204,848

Abstract

This is the final report on the research performed under NASA Glen grant NASA/NAG-3-1975 concerning feedback control of the Pratt & Whitney (PW) STF 952, a twin spool, mixed flow, after burning turbofan engine. The research focussed on the design of linear and gain-scheduled, multivariable inner-loop controllers for the PW turbofan engine using H_∞ and linear, parameter-varying (LPV) control techniques. The nonlinear turbofan engine simulation was provided by Pratt & Whitney within the NASA ROCETS simulation software environment. ROCETS was used to generate linearized models of the turbofan engine for control design and analysis as well as the simulation environment to evaluate the performance and robustness of the controllers. Comparison between the $\mathcal{H}_\infty$ and LPV controllers are made with the baseline multivariable controller and developed by Pratt & Whitney engineers included in the ROCETS simulation. Simulation results indicate that $\mathcal{H}_\infty$ and LPV techniques effectively achieve desired response characteristics with minimal cross coupling between commanded values and are very robust to unmodeled dynamics and sensor noise.

Chapter 1

Introduction

The design of traditional gain-scheduled, inner-loop controllers for a turbofan engine takes significant time and can not guarantee performance and stability of the closed-loop system across the operating envelope [10]. Linear parameter-varying (LPV) control methodologies can however, guarantee stability and performance over the entire operating envelope, making them a natural approach for inner-loop control of turbofan engines. This report documents the application of LPV control design techniques to a PW turbofan engine. Pratt & Whitney and NASA Glenn at Lewis Field Research Center provided the University of Minnesota with a transient turbofan engine simulation (ROCETS) to use as a test bed for implementing LPV control designs. The baseline multivariable control algorithm in ROCETS was implemented as many interconnected subroutines rather than as a single integrated subroutine. This was not conducive to implementation and simulation of candidate controllers, and necessitated changes to the controller software implementation in ROCETS. This report discusses the revisions required to implement state-space and LPV controllers, the design of $\mathcal{H}_\infty$ and LPV controllers, and the results obtained from the implementation of the controllers in the Pratt & Whitney engine simulation.

1.1 Accomplishments

This grant partially supported the turbofan engine control research of Professors Gary Balas and William Garrard, a post-doctoral researcher, Dr. Greg Wolodkin, two graduate students, Jack Ryan and Dr. Jeff Barker, and two undergraduate researchers, Chris Mitchell and Edward Harper. Dr. Wolodkin currently works at the Mathworks and was one of their lead engineers in the advanced controls area. He has since moved within the Mathworks and

is currently responsible for the core Matlab product. Dr. Barker graduated with the Ph.D. degree in 1999 and is employed by Boeing Phantom Works in St. Louis. Jack Ryan wrote his Master's project report on the turbofan engine research and is currently employed by NASA Dryden Flight Research Center as a flight simulation engineer. (Note that Chapters 2, 3 and 5 of this report are based, in part, on his Master's report.) Chris Mitchell is currently a Ph.D. student in the Aerospace Engineering and Mechanics department at the University of Minnesota in fluid mechanics and Edward Harper is currently working as an engineer in industry.

The following are a list of the technical accomplishments achieved with the support of this grant.

- Learned to use the NASA Rocket Engine Transient Simulation (ROCETS) system to simulate the turbofan engine model provided by Pratt and Whitney (PW).
- Development of a linear, parameter-varying (LPV) model of the PW turbofan engine for control design.
- Developed and integrated Fortran subroutines to implement linear and LPV state-space controllers into the ROCETS nonlinear simulation.
- Successfully back engineered the baseline PW multivariable controller.
- Synthesized robust, linear multivariable $\mathcal{H}_\infty$ controllers and gain-scheduled LPV controller for the turbofan engine.
- Successfully implemented these controllers in the ROCETS nonlinear simulation.
- Achieved performance robustness of the non-rate and rate bounded LPV controllers. The LPV controllers were scheduled on a lagged measurement of power code. These controllers performed well for a variety of environmental conditions, through the flight envelope with and without noisy sensor measurements.

Three papers and a Masters project report were written on the synthesis of controllers for turbofan engine during the length of this contract. A fourth paper is in preparation based on the latest results of the gain-scheduled LPV inner-loop controllers for the PW turbofan engine model. The technical monitor for this grant was Jonathan Litt, NASA Glenn Research Center.

- G. Wolodkin, G.J. Balas, W.L. Garrard, "Application of parameter-dependent robust control synthesis to turbofan engines," *AIAA Journal of Guidance, Dynamics and Control* , vol. 22, no. 6, 1999, pp. 833-838.

- G. Wolodkin, G.J. Balas, W.L. Garrard, "Application of Parameter-Dependent Robust Control Synthesis to Turbofan Engines," *36th AIAA Aerospace Sciences Meeting*, Reno, NV, January, 1998, AIAA-98-0973.

- G.J. Balas, J. Ryan, J.Y. Shin, W.L. Garrard, "A New Technique for Design of Controllers for Turbofan Engines," *34th Joint Propulsion Conference*, Session 95-ASME-21, Cleveland, OH, July, 1998, AIAA-98-3751.

Chapter 2

Engine Simulation

The Pratt & Whitney STF 952 (Figure 2.1), a twin spool, mixed flow, after burning turbofan engine, is used as the example application in this study. It features a highly loaded, three stage fan, a four stage high pressure compressor, an axially staged triangular alignment combuster, and an advanced high pressure turbine and low pressure turbine. For later reference, the fan inlet is identified as station 2, the high pressure turbine inlet as station 4, and the turbine exit guide vane as station 6. The pressures at each station are designated as P2, P4, and P6 respectively. A diagram of the engine and the actuator and sensor locations is given in Figure 2.1.

The STF 952 engine is modeled using the NASA Rocket Engine Transient Simulation (ROCETS) software. The ROCETS model of the STF 952 engine is fully nonlinear and it uses a multivariable integration routine based on a modified Newton-Raphson technique to calculate transient responses and steady-state balance. The linearized models of the STF 952 engine are all generated using ROCETS as are all nonlinear closed-loop simulations. Through out this report, we will denote the PW STF 952 turbofan engine as the "engine" or "turbofan engine" in the text.

The engine dynamics vary with thrust request or "power code," temperature and external air pressure. Air pressure and temperature vary with altitude as well as weather. The engine power code varies from 3,000 to 30,000 which corresponds to near idle to military power. Initially, the altitude is set at sea level, 0K ft, a speed of zero Mach and standard atmosphere. The objective of the control system is to accurately track inner-loop commands to the engine.

A turbofan engine transient performance computer model of the SCIP Engine (STF 952A) was configured using the NASA Rocket Engine Transient Simulation (ROCETS) system.

ROCETS interfaces modular components to generate a full engine simulation in which a run processor reads and interprets input to run particular experiments. A multi-variable modified Newton-Raphson technique is used for transient integrations and steady-state balances [1]

Schedules of desired thrust level, altitude, Mach number, or other variables can be input to the simulation. The simulation model of engine dynamics generate the necessary overall pressure ratio ($OPR = P4/P2$), engine pressure ratio ($EPR = P6/P2$) and high pressure compressor spool speed (N2) requests to meet the scheduled variables. These requests (OPRREQ, EPRREQ, N2REQ) are fed to the inner-loop control system which generates the required primary burner fuel flow (WFPRIB), high pressure compressor normalized variable vane angle (VANEHPC), and convergent throat area (AREANOZL) commands to achieve the requests. This control inner-loop is the focus of this report.

The current multivariable controller in the turbofan simulation, used as a baseline to compare with designed controllers, operates in two modes: start mode and nominal multivariable mode. The start mode is used to transition from light off fuel flow, to nominal multivariable control mode. The nominal multivariable mode uses a controller scheduled on total corrected airflow [1]. This paper focuses on the design of a controller for the nominal multivariable mode. The baseline controller is used during the initialization part of the simulation.

The baseline control interconnection (Figure 2.2) has an two loops.

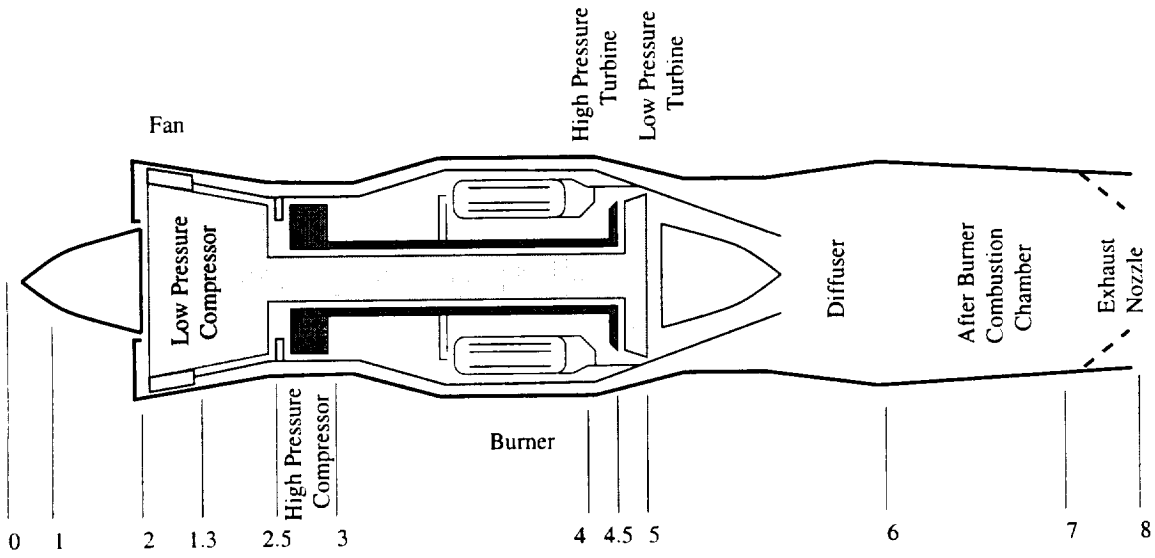

Figure 2.1: STF 952 Turbofan Engine

The inner control loop consists of the state space matrix K and the gain matrix S. The K

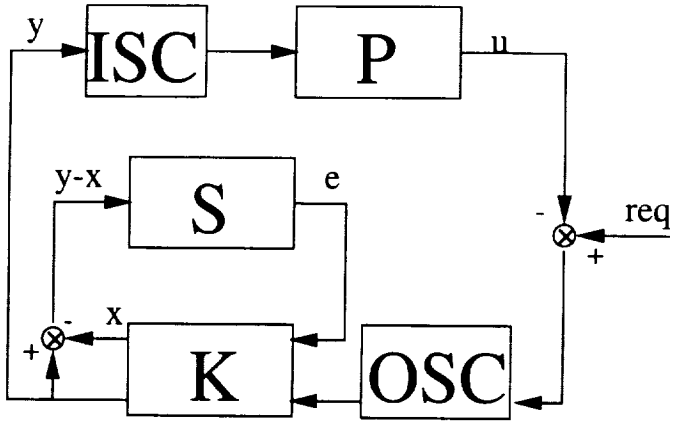

Figure 2.2: Interconnection of Engine Model and Baseline PW Controller

matrix has the form

$$K = \left[\begin{array}{ccccc|ccccccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & B_{1,1} & B_{1,2} & B_{1,3} \\ A_{2,1} & 1-A_{2,1} & A_{2,3} & -A_{2,3} & 0 & 0 & 0 & 0 & B_{2,1} & B_{2,2} & B_{2,3} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & B_{3,1} & B_{3,2} & B_{3,3} \\ A_{4,1} & -A_{4,1} & A_{4,3} & 1-A_{4,3} & 0 & 0 & 0 & 0 & B_{4,1} & B_{4,2} & B_{4,3} \\ A_{5,1} & -A_{5,1} & A_{5,3} & -A_{5,3} & 1 & 0 & 0 & 0 & B_{5,1} & B_{5,2} & B_{5,3} \\ \hline 0 & 1 & 0 & 0 & 0 & S_{1,1} & S_{1,2} & S_{1,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & S_{2,1} & S_{2,2} & S_{2,3} & 0 & 0 & 0 \\ A_{5,1} & -A_{5,1} & A_{5,3} & -A_{5,3} & 1 & S_{3,1} & S_{3,2} & S_{3,3} & 0 & 0 & 0 \end{array}\right] \tag{2.1}$$

where

$$\begin{array}{lll} S_{1,1} = B_{1,1} + D_{1,1} & S_{1,2} = B_{1,2} + D_{1,2} & S_{1,3} = B_{1,3} + D_{1,3} \\ S_{2,1} = B_{3,1} + D_{2,1} & S_{2,2} = B_{3,2} + D_{2,2} & S_{2,3} = B_{3,3} + D_{2,3} \\ S_{3,1} = B_{5,1} + D_{3,1} & S_{3,2} = B_{5,2} + D_{3,2} & S_{3,3} = B_{5,3} + D_{3,3} \end{array} \tag{2.2}$$

The values of $A_{i,j}$, $B_{i,j}$, $D_{i,j}$ are determined via a look-up table scheduled on operating condition. The states of the controller are: gas generator fuel flow (x_1), precursor to the normalized gas generator fuel flow (x_2), normalized flow parameter(x_3), precursor to the normalized flow parameter(x_4), and normalized compressor variable vane(x_5).

The inputs to K are

$$u = \left[\begin{array}{ccc} u_1 & u_2 & u_3 \end{array}\right]^T \tag{2.3}$$

and

$$e = \left[\begin{array}{ccc} e_1 & e_2 & e_3 \end{array}\right]^T \tag{2.4}$$

u consists of normalized overall pressure ratio error (u_1), normalized engine pressure ratio error (u_2), and normalized rotor speed error (u_3). The vector e consists of overall pressure ratio error from the last pass through the loop (e_1), engine pressure ratio error from the last pass through the loop (e_2), and high rotor speed error from the last pass through the loop (e_3).

The outputs from the controller are

$$x = \begin{bmatrix} x_1 & x_4 & x_5 \end{bmatrix}^T \tag{2.5}$$

and

$$y = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix}^T \tag{2.6}$$

y_1 is normalized fuel flow request, y_2 is normalized flow parameter request, and y_3 is normalized compressor variable vane request.

The S matrix has the form

$$\frac{1}{DETGN} \begin{bmatrix} S_{2,2}S_{3,3} - S_{3,2}S_{2,3} & S_{3,2}S_{1,3} - S_{1,2}S_{3,3} & S_{1,2}S_{2,3} - S_{2,2}S_{1,3} \\ S_{3,1}S_{2,3} - S_{2,1}S_{3,3} & S_{1,1}S_{3,3} - S_{3,1}S_{1,3} & S_{2,1}S_{1,3} - S_{1,1}S_{2,3} \\ S_{2,1}S_{3,2} - S_{3,1}S_{2,2} & S_{3,1}S_{1,2} - S_{1,1}S_{2,1} & S_{1,1}S_{2,2} - S_{2,1}S_{1,2} \end{bmatrix} \tag{2.7}$$

where

$$DETGN = S_{1,1}S_{2,2}S_{3,3} + S_{1,2}S_{2,3}S_{3,1} + S_{1,3}S_{2,1}S_{3,2} - S_{3,1}S_{2,2}S_{1,3} - S_{3,2}S_{2,3}S_{1,1} - S_{3,3}S_{2,1}S_{1,2} \tag{2.8}$$

It has inputs $y_1 - x_1$, $y_2 - x_4$,and $y_3 - x_5$, and outputs e_1, e_2, and e_3. Together, K and S form the baseline controller for inner-loop control of the PW turbofan engine.

The closed-loop system consists of the plant model (P), two constant gain matrices (OSC and ISC), and the controller. The outer-loop control generates the desired values of overall pressure ratio (OPRREQ), engine pressure ratio (EPRREQ), and high speed rotor request (N2REQ) based on environmental conditions, power code and the baseline closed-loop engine dynamic response.

The inputs to the plant, P in Figure 2.2, are WFPRIB, VANEHPC, and AREANOZL. The plant outputs are P2, P4, P6, and N2. The matrix OSC normalizes its inputs from overall pressure ratio error (OPRREQ-OPR), engine pressure ratio error (EPRREQ-EPR) and high rotor speed error(N2REQ-N2) to the normalized parameters u_1,u_2, and u_3. The matrix ISC dimensionalizes its inputs from y_1, y_2, and y_3 to WFPRIB, AREANOZL, and VANEHPC. The ISC gain matrix is

$$\begin{bmatrix} 7 & 0 & 0 \\ 0 & 807 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

and the OSC gain matrix is

$$\begin{bmatrix} \frac{0.0320491}{14.55} & 0 & 0 \\ 0 & \frac{0.1599723}{14.55} & 0 \\ 0 & 0 & 0.000055528 \end{bmatrix} \tag{2.10}$$

2.1 Controller Subroutines

To implement state-space multivariable controllers, changes were required to the ROCETS nonlinear simulation of the turbofan engine model. These modifications allow 3 input, 3 output multivarible controllers of state order up to 21 to be read in and replace the baseline Pratt & Whitney controller. This allowed controllers to be easily tested in the simulation without rewriting or recompiling the ROCETS simulation. Changes were also made so that the simulation uses the Pratt & Whitney controller in the start up mode until it has reached the multivariable control mode. This eliminates engine start-up issues in the design of $\mathcal{H}_\infty$ and LPV controllers.

The new code reads in the synthesized controller at the beginning of each simulation. If the controller has less than 21 states, it is padded with zeros so that the multiplication routine only need work with a constant size control matrix. Figure 2.3 shows the implementation of a five state controller in the control algorithm. The usual $ABCD$ five state control matrix is located in the bottom right corner. Zeros fill the remaining entries keeping x_1 through x_{16} constant zeros. Details of the algorithm are presented in Appendix B.

The ROCETS state-space controller implementation was tested with the Pratt & Whitney baseline controller and the results were compared with the original simulation of the PW baseline controller implementation. This was necessary to demonstrate our understanding of the existing code and our ability to replace the control algorithms with out jeopardizing the overall engine simulation. The baseline simulation matched well with the new implementation verifying the state-space algorithms. Figure 2.4 compares the Pratt & Whitney control inputs and outputs with our implementation of the Pratt & Whitney controller. Pratt & Whitney's N2 follows its request ramping up after the step input while our implementation's N2 follows its request which does not ramp up. The difference is due to the reference command request generating algorithm which is not part of this research project. EPR, OPR and N2 commands are treated as exogenous inputs in the control problem since we do not have direct control of them. The differences in VANEHPC are directly due to the variations in the N2 commands. The small variations in the other channels are not significant.

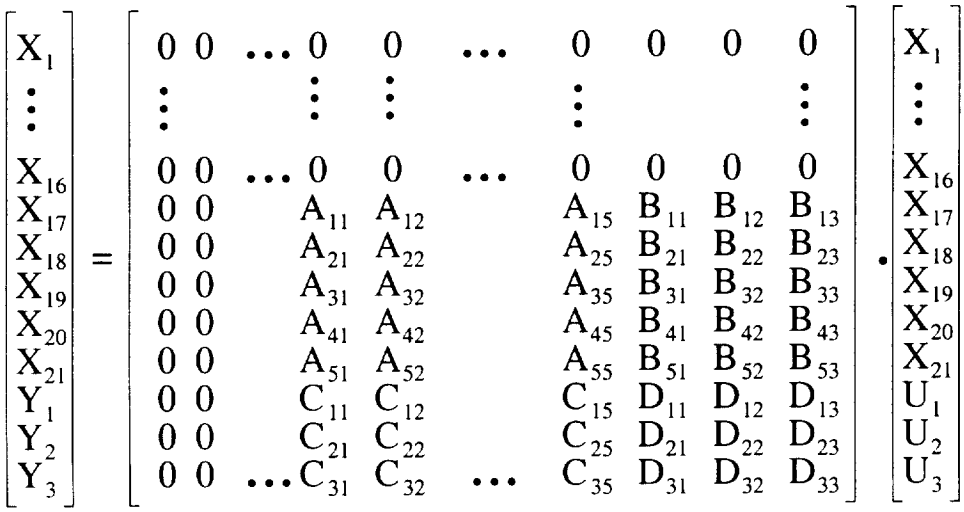

Figure 2.3: Five State Controller in 21 State Framework

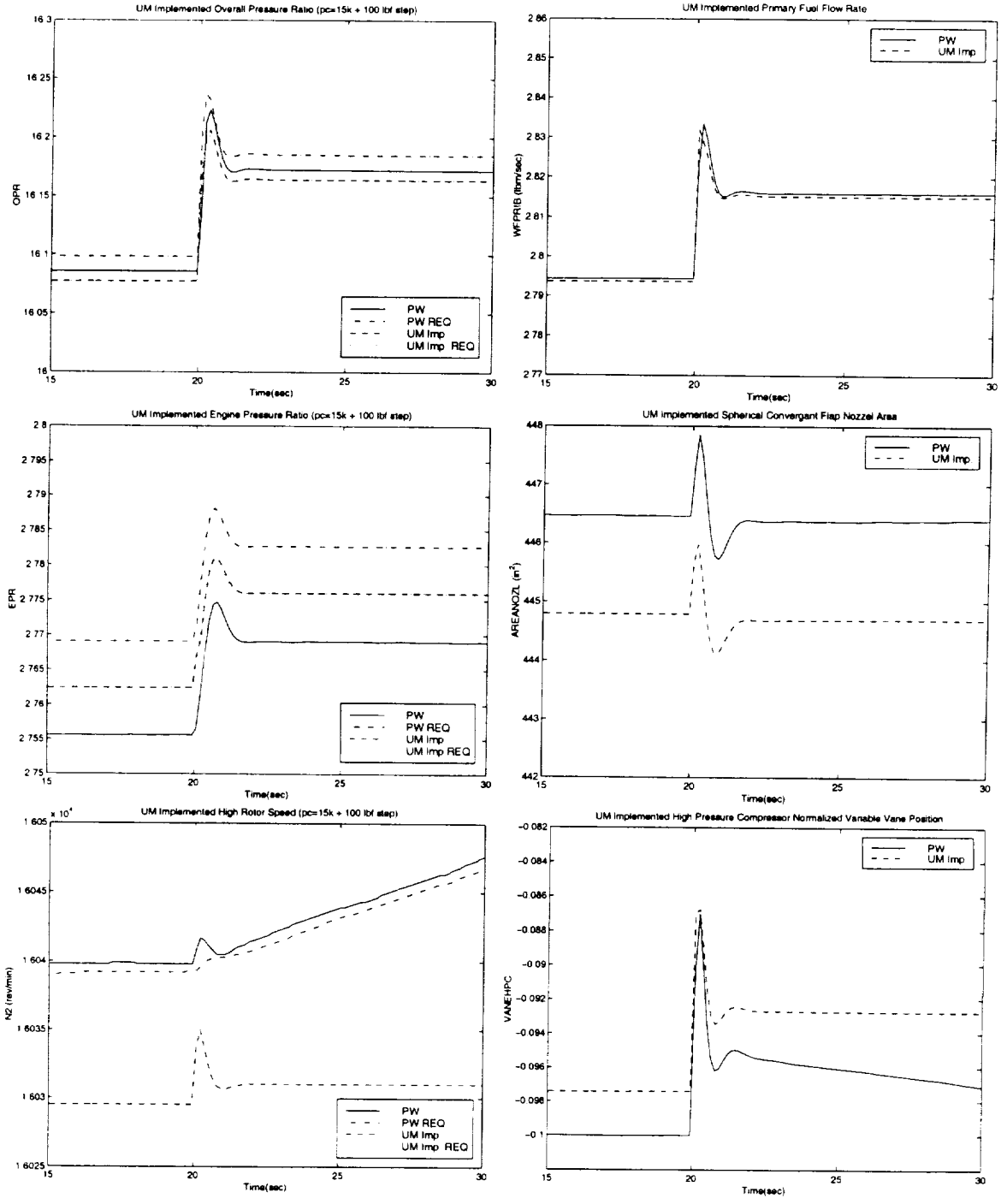

Figure 2.4: UM implementation of PW Baseline Inner-loop Controller

2.2 Engine Linearized Models

Ten Jacobian linearized plant models were generated using the nonlinear simulation between 3000 and 30000 lbf thrust at 3000 lbf thrust intervals and different altitudes. Each model was generated for a given altitude, zero Mach, zero angle of attack, standard atmosphere, zero side slip, and 14.696 psi static pressure. The Jacobian linearizations have 11 states, three inputs, and three outputs. The states are identified in Table 2.1, the inputs in Table 2.2,and the outputs in Table 2.3. The simulation's input file used to generate the linear plants along with Matlab utilities to move the generated plants into Matlab files are provided in Appendix A.3 and A.A.4.4 respectively.

State	Description
SN1	Low Rotor Physical Speed (rpm)
TMCHPC	High pressure compressor case lumped metal temperature (deg R)
TMBHPC	High pressure compressor blade lumped metal temperature (deg R)
TMRHPC	High pressure compressor rotor lumped metal temperature (deg R)
TMCHPT	High pressure turbine case lumped metal temperature (deg R)
TMBHPT	High pressure turbine blade lumped metal temperature (deg R)
TMRHPT	High pressure turbine rotor lumped metal temperature (deg R)
TMCLPT	Low pressure turbine case lumped metal temperature (deg R)
TMBLPT	Low pressure turbine blade lumped metal temperature (deg R)
TMRLPT	Low pressure turbine rotor lumped metal temperature (deg R)
TMILBN	Main burner liner metal temperature (deg R)

Table 2.1: Linear Plant States

Input	Description
WFPRIB	Primary Fuel Flow rate (lbm/sec)
VANEHPC	High pressure compressor normalized variable vane position
AREANOZL	Spherical Convergent flap nozzle physical area (in^2)

Table 2.2: Linear Plant Inputs

Figure 2.5 shows the magnitude and phase of the linearized plants. The dotted line represents the 3000 lbf model, the solid 15000 lbf model, and the dash-dot line 30000 lbf model. All other power codes are represented by the dash lines.

The linearized engine models are used for the $\mathcal{H}_\infty$ and LPV control designs. The plots in Figure 2.5 indicate that the state order of these models may be reduced. Balanced realization

Output	Description
P2	Engine face total pressure (psi)
P4	Primary burner exit total pressure (psi)
P6	Turbine exit guide vane exit total pressure (psi)
N2	High rotor physical speed (rpm)

Table 2.3: Linear Plant Outputs

model reduction will be used to reduce the plant state order. Since the LPV design model requires all states to have the same meaning, the same balancing transformation matrix must be used for all plant models. The transformation matrix at the 12,000 power code was selected to balance the models. After balancing, all engine models were truncated to three states with no effect on the plant dynamics.

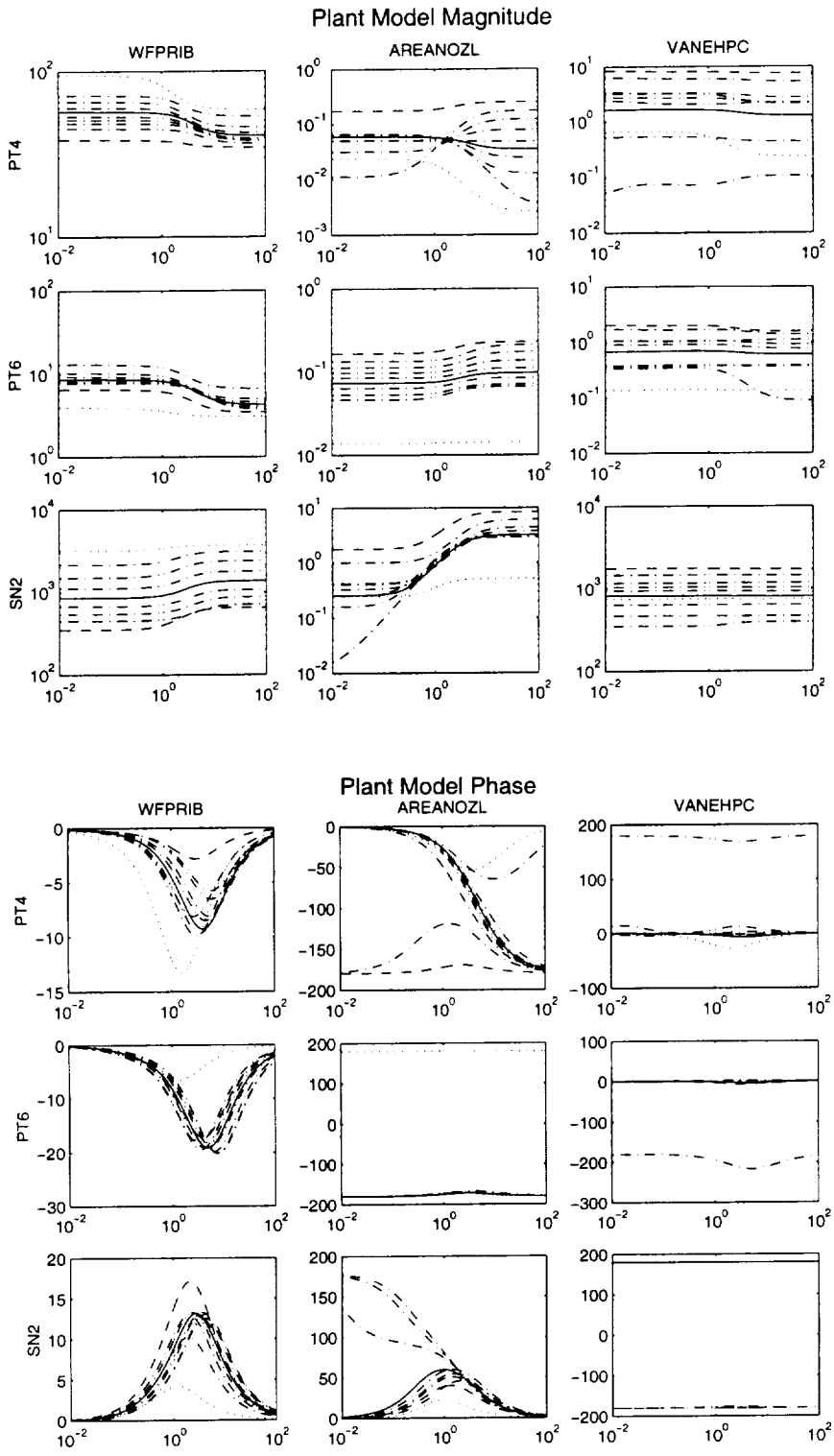

Figure 2.5: Magnitude (top) and Phase (bottom) of Jacobian Linearizations at Sea Level and Standard Day Atmosphere.

Chapter 3

$\mathcal{H}_\infty$ Controllers

$\mathcal{H}_\infty$ control design techniques are used to synthesize controllers for the PW turbofan at ten power code operating points. Jacobian linearizations of the turbofan engine were generated at power codes from 3000 to 30000 lbf thrust in 3000 lbf increments. The reader is referred to references [18, 13, 16] for details on $\mathcal{H}_\infty$ control theory and its application. The control objectives were to achieve good tracking of OPR, EPR, and N2 commands, decoupled command response, disturbance rejection below 2 rad/sec and a 10 rad/sec bandwidth and robustness to modeling error, sensor noise and changing environmental conditions. The $\mathcal{H}_\infty$ controllers also must respect the physical limits on the actuator deflections and rates. The controllers are designed using a model matching problem formulation in the $\mathcal{H}_\infty$ framework (see Figure 3.1). We desired the engine to respond as three single-input/single-output (SISO) systems with no off-diagonal coupling. P_{mod} represents the decoupled system. Differences between this desired model and the true plant are penalized via the weight W_p.

The disturbance vector d_1 represents customer demands, whose values we wish the plant output to track. Specifically these commands correspond to desired normalized overall engine pressure ratio, normalized engine pressure ratio and normalized high rotor speed. The disturbance vector d_2 is used to represent both noise and input uncertainty in the problem formulation to which the controller should be robust. The input uncertainty is mapped to the output of the plant model to reduce the number of states required to define the open-loop control design interconnection.

By keeping the error vector e_1 small, good tracking of d_1 is ensured. The role of e_2 is to penalize control effort, in terms of actuator magnitudes as well as actuator rates. The controller sees y_c as its input measurements, and generates u_c as its output.

The actuators are modeled as P_{act}, a diagonal augmentation of unity gain first-order lags,

$$P_{act} = \frac{100}{s+100} I_{3\times 3} \tag{3.1}$$

The actuator model outputs actuator positions u and actuator rates $\dot{u}$, to allow actuator rates to be penalized in the control design.

The model-matching block P_{mod} was based on previous work in Reference [2]. It is desired that the engine response to OPR and EPR request follow a second order model with natural frequency of 10 rad/sec and damping of 0.65. The engine response to N2 request should follow a second order model with natural frequency of 2.5 rad/sec and damping of 0.65.

$$p_{mod_i} = \frac{\omega_i^2}{s^2 + 2\xi\omega_i + \omega_i^2}, \qquad P_{model} = \begin{bmatrix} p_{mod_1} & 0 & 0 \\ 0 & p_{mod_1} & 0 \\ 0 & 0 & p_{mod_2} \end{bmatrix}$$

Here $\omega_1 = 10$ rad/s, $\omega_2 = 2.5$ rad/s, and $\xi = 0.65$.

The input weight W_i is a constant weighting used to normalize the inputs. The input weight

$$W_i = \begin{bmatrix} 0.08 & 0 & 0 \\ 0 & 0.06 & 0 \\ 0 & 0 & 0.25 \end{bmatrix}$$

in these designs.

The disturbance W_d is used to model actuator errors as well as limit the bandwidth of control effort by ramping up at high frequency.

$$W_d = 0.0005 \frac{\frac{1}{0.1}s + 1}{\frac{1}{30000}s + 1} I_{3\times 3},$$

The control weights W_c are selected to be constant. Here we have chosen to penalize the normalized actuator movement larger than unity, as well as actuator rates,

$$W_c = \begin{bmatrix} 0.05 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.02 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.05 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.05 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.005 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.05 \end{bmatrix}$$

The performance weight W_p penalizes the difference between the desired and the actual closed-loop response of the turbofan engine. The larger the magnitude of W_p, the smaller the allowable difference between desired and actual output. In the initial designs, W_p is a first order low-pass weight given by

$$W_p = \begin{bmatrix} 500\frac{\frac{1}{1500}s+1}{\frac{1}{25}s+1} & 0 & 0 \\ 0 & 300\frac{\frac{1}{1500}s+1}{\frac{1}{1.5}s+1} & 0 \\ 0 & 0 & 400\frac{\frac{1}{1500}s+1}{\frac{1}{2}s+1} \end{bmatrix}$$

This ensures good tracking of the response models in the bandwidth 1–10 rad/s, with little or no DC error. At low frequency (below 0.1 rad/s) the W_p weight on the OPR corresponds to a DC errors of $\frac{1}{500}$ or 0.2% tracking error. The EPR tracking error at DC is $\frac{1}{300}$ or 0.33% and the N2 tracking error at DC can be $\frac{1}{400}$ or 0.25%. At high frequency, the mismatch between actual and desired output is not penalized.

The matrix OSC normalizes its inputs from overall pressure ratio error (OPRREQ-OPR), engine pressure ratio error (EPRREQ-EPR) and high rotor speed error(N2REQ-N2).

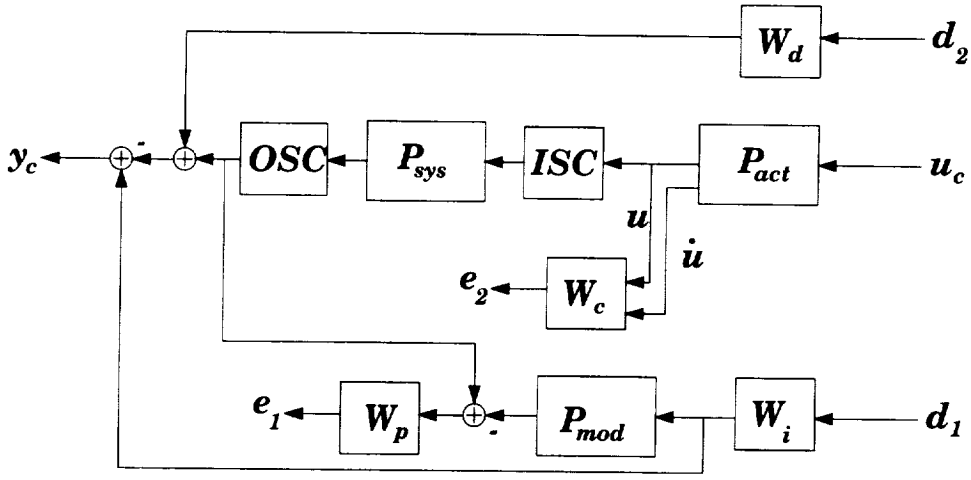

Figure 3.1: $\mathcal{H}_\infty$ control design interconnection

The open-loop interconnection shown in Figure 3.1 has 18 states. Hence the resulting controllers had 18 states. Eight of the ten $\mathcal{H}_\infty$ point designs at the power codes between 3K and 30K achieved an $\mathcal{H}_\infty$ norm less than 1.62 and all ten were under 2.4. They were first tested in linear simulations and then in the nonlinear ROCETS simulation for small step commands. While the linear simulations were satisfactory, the nonlinear simulation resulted in highly oscillatory reference commands. Recall that the reference commands are generated by an outer-loop controller which has not been modified from the original PW baseline design.

Figure 3.2 and 3.3, show the ROCETS simulation time response with a linear $\mathcal{H}_\infty$ controller implement and a step command from 15000 lbf to 15100 lbf. The simulation-generated commands of OPR, EPR, and N2 are all oscillatory resulting in oscillatory responses. Controllers designed for the other nine power codes had similar results.

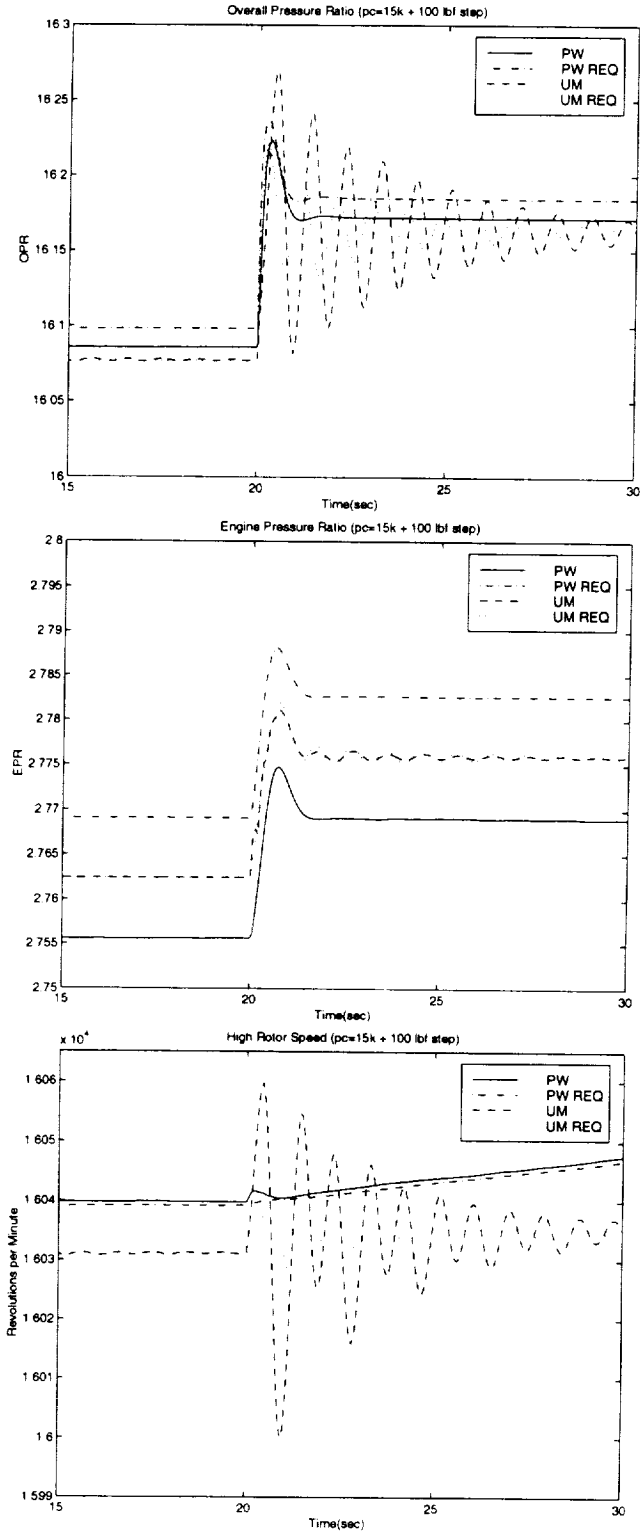

Figure 3.2: Controller Inputs of $\mathcal{H}_\infty$ Controlled System

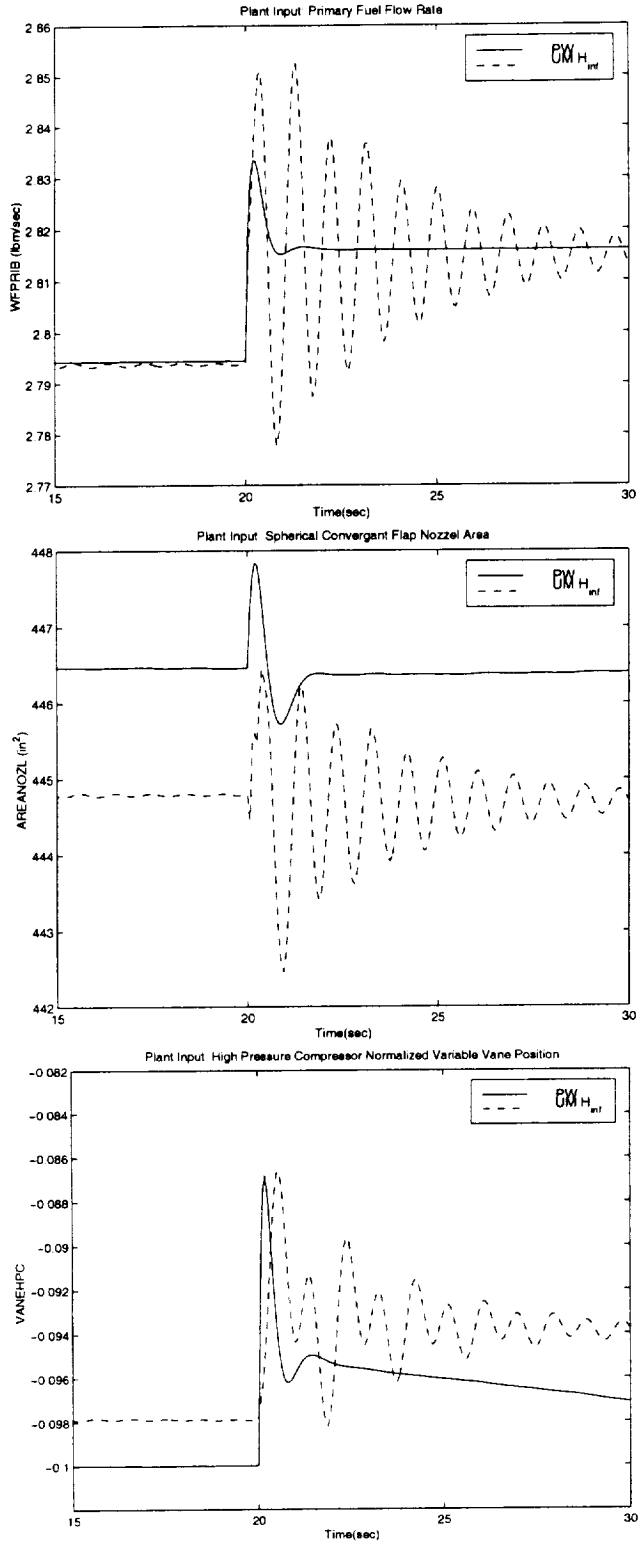

Figure 3.3: Plant Inputs of $\mathcal{H}_\infty$ Controlled System

3.1 Identification of Linear Engine Models

The good performance of the $\mathcal{H}_\infty$ controllers on the linear engine models and the poor performance of these controllers in the nonlinear simulation required investigation. The first hypothesis was an error in the implementation of the linear state-space controller subroutine in ROCETS. After testing, it was found that this was not the case. Therefore, the problem must have been related to the inaccuracy of the linearized engine model relative to the nonlinear engine simulation. This could be due to several reasons. The first is our poor understanding of the ROCETS simulation code. Hence we may have incorrectly linearized the engine. Similarly, the ROCETS linearization may only include engine dynamics and therefore all the actuator and sensor dynamics, as well as filtering and computational delays need to be included in the linear engine model as well. The linear models used in the controller synthesis did not capture all the nonlinear plant dynamics. These unmodelled dynamics, we conjectured, were causing the oscillations. The linear and nonlinear responses of the plant to single channel inputs were therefore examined.

The nonlinear ROCETS simulation needed to be modified since it was not designed for such an examination. By setting two of the three controller outputs to zero, the remaining plant input could be set to a variable frequency sinusoidal. Figure 3.4 shows the responses to the single sinusoidal inputs. The nonlinear simulation response is represented by the solid lines, and the linear models response by the dashed lines. In each set of plots, the top left plot displays the active input signal: WFPRIB in the first, AREANOZL in the second, and VANEHPC in the third. The remaining plots display the signal responses of OPR, EPR, and N2. We can see that the linear and nonlinear models do not match exactly. In particular, the nonlinear responses of N2 are smoother than their linear counterparts. They respond slower and subsequently lack the sharp overshoot. After examining all the controller inputs and outputs, to better match the nonlinear engine simulation: the actuator model was modified, a sensor model added to the rotor speed sensor, N2, and a lag filter was included in the high speed pressure compressor vane actuator, to help slow down the response of N2 and compensate for the differences between the linear time response and nonlinear simulation. The small signal simulations of the modified linear engine models matched the nonlinear engine simulations well across power code at sea level and standard atmosphere. The new actuator model was chosen as

$$P_{act} = \begin{bmatrix} \frac{20}{s+20} & 0 & 0 \\ 0 & \frac{20}{s+20} & 0 \\ 0 & 0 & \frac{15}{s+15} \\ \frac{20s}{s+20} & 0 & 0 \\ 0 & \frac{20s}{s+20} & 0 \\ 0 & 0 & \frac{15s}{s+15} \end{bmatrix}$$

A sensor model was added to the $N2$ measurement,

$$sen = \frac{\frac{1}{20}s + 1}{\frac{1}{4}s + 1}$$

and a lag filter was added to the $N2$ command channel,

$$lag = \frac{\frac{1}{96}s + 1}{\frac{1}{12}s + 1}$$

The modified linear engine models frequency responses are shown in Figure 3.5. The N2 phase change is apparent as is the VANEHPC input smooth roll off at high frequency. (See Figure 2.5 for comparison.). The modified linear models of the engine are used to synthesize all the subsequent $\mathcal{H}_\infty$ and LPV control designs.

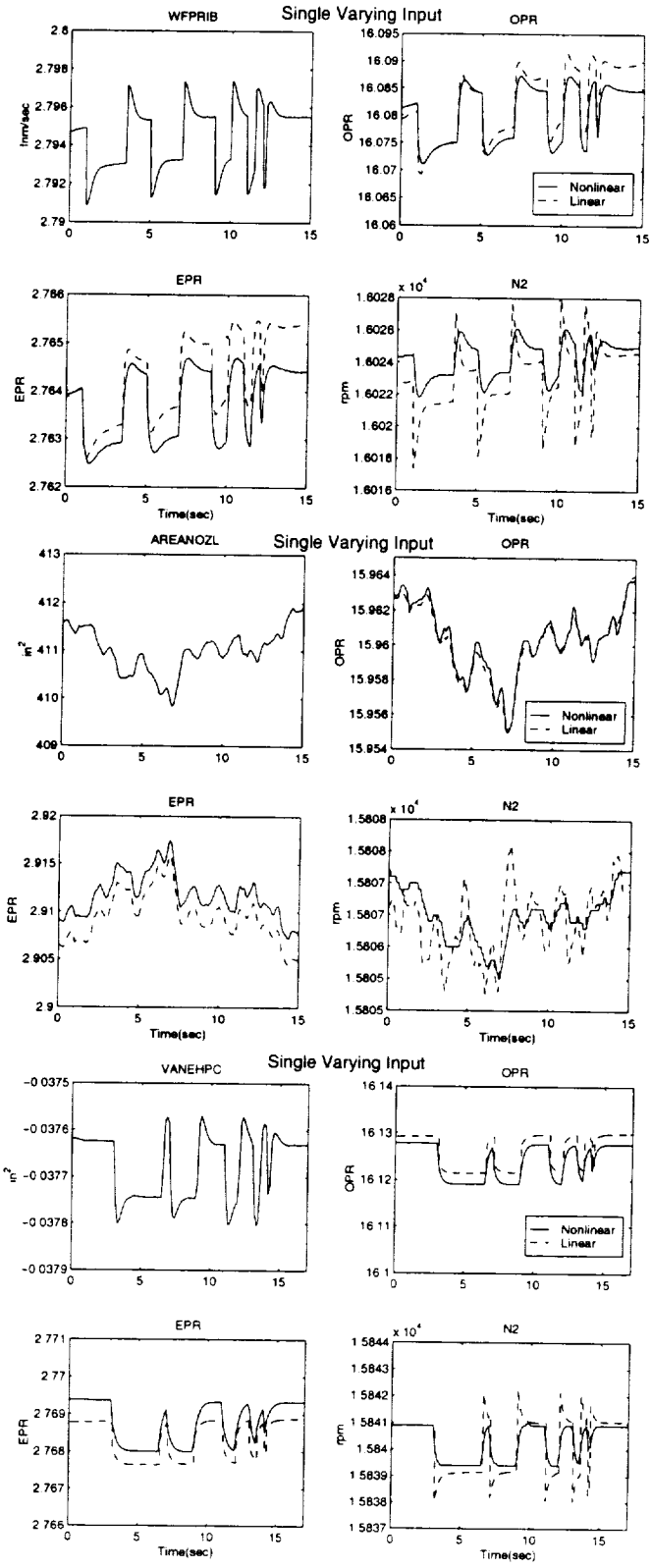

Figure 3.4: Single Input Responses

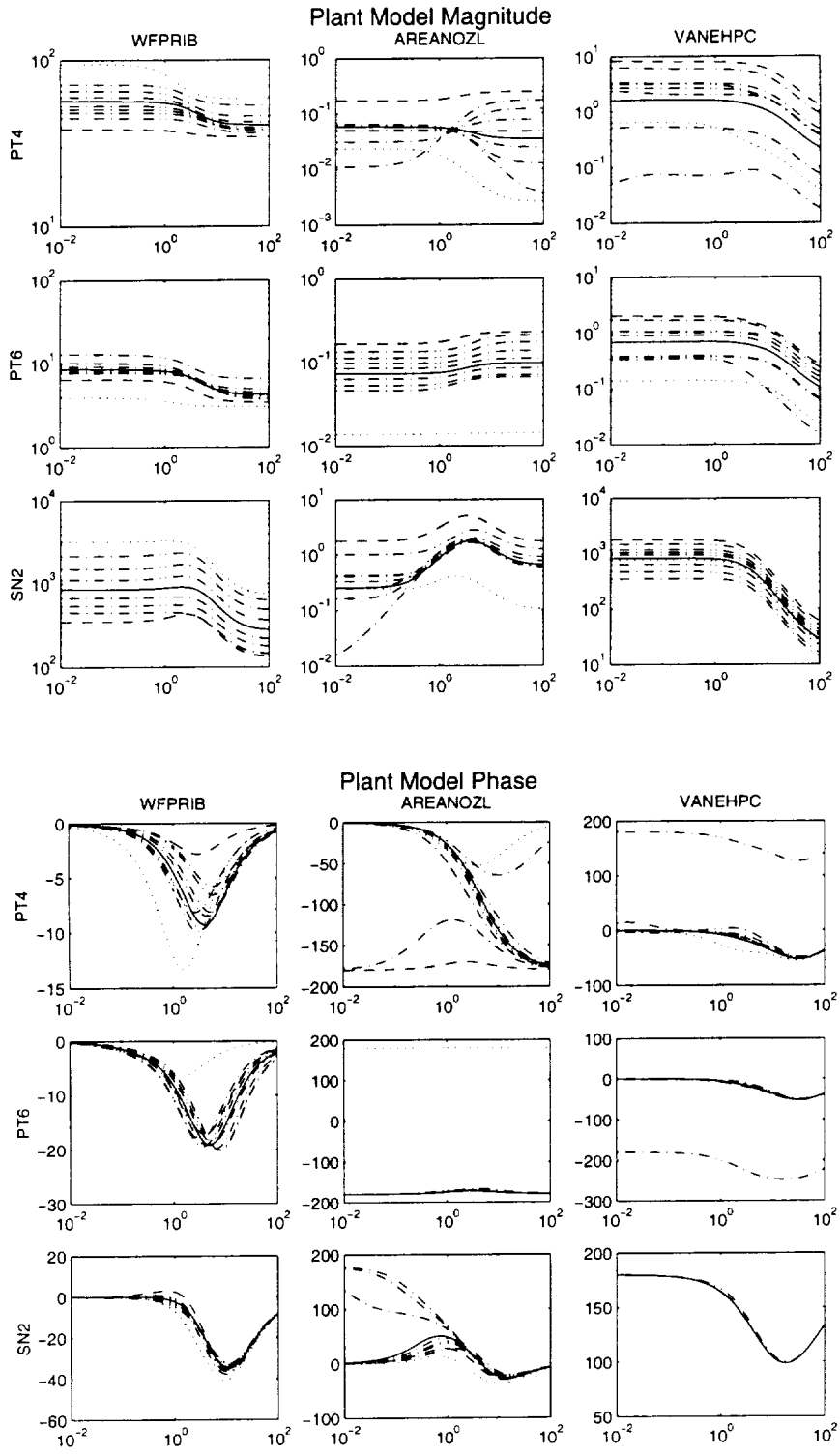

Figure 3.5: Magnitude (top) and Phase (bottom) of Modified Linear Engine Models with Lag, Actuator and Sensor Models

3.2 Redesigned $\mathcal{H}_\infty$ Controllers

The $\mathcal{H}_\infty$ controllers were re-synthesized using the same weights as used in the original $\mathcal{H}_\infty$ designs but with the new lag, sensor model and actuator model. The redesigned controllers had 20 states and achieved a $\mathcal{H}_\infty$ norm of 1.54. Time responses in the nonlinear ROCETS simulation of the new 15000 lbf thrust system to a 100 lbf step are shown in Figure 3.7. The oscillations in the reference commands have been eliminated. The $\mathcal{H}_\infty$ controllers achieved better tracking than the baseline controller in OPR and EPR, with comparable overshoot. Note that with the $\mathcal{H}_\infty$ controller implemented in the nonlinear engine simulation, the response of N2 exhibits more overshoot but better tracking of the N2 command as compared with its baseline counterpart.

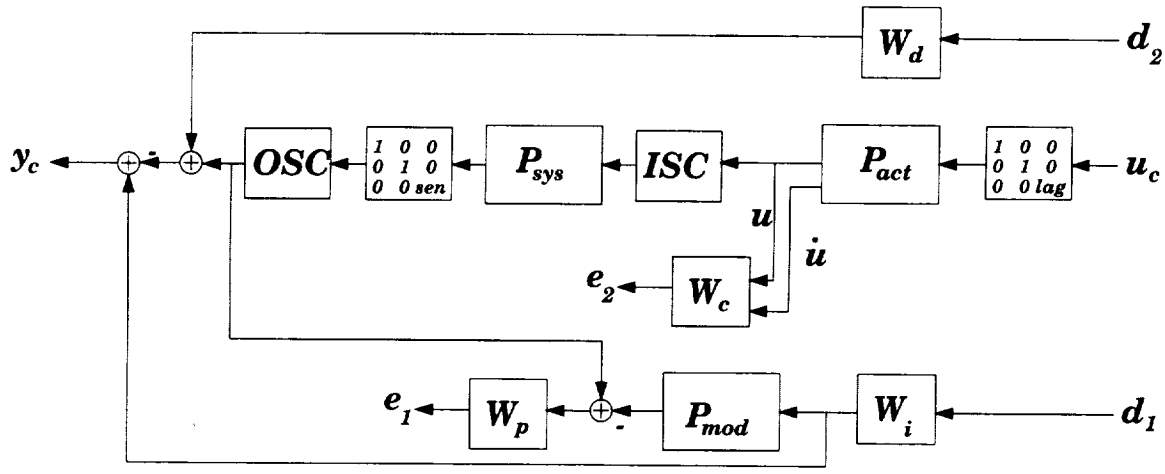

Figure 3.6: $\mathcal{H}_\infty$ control design interconnection

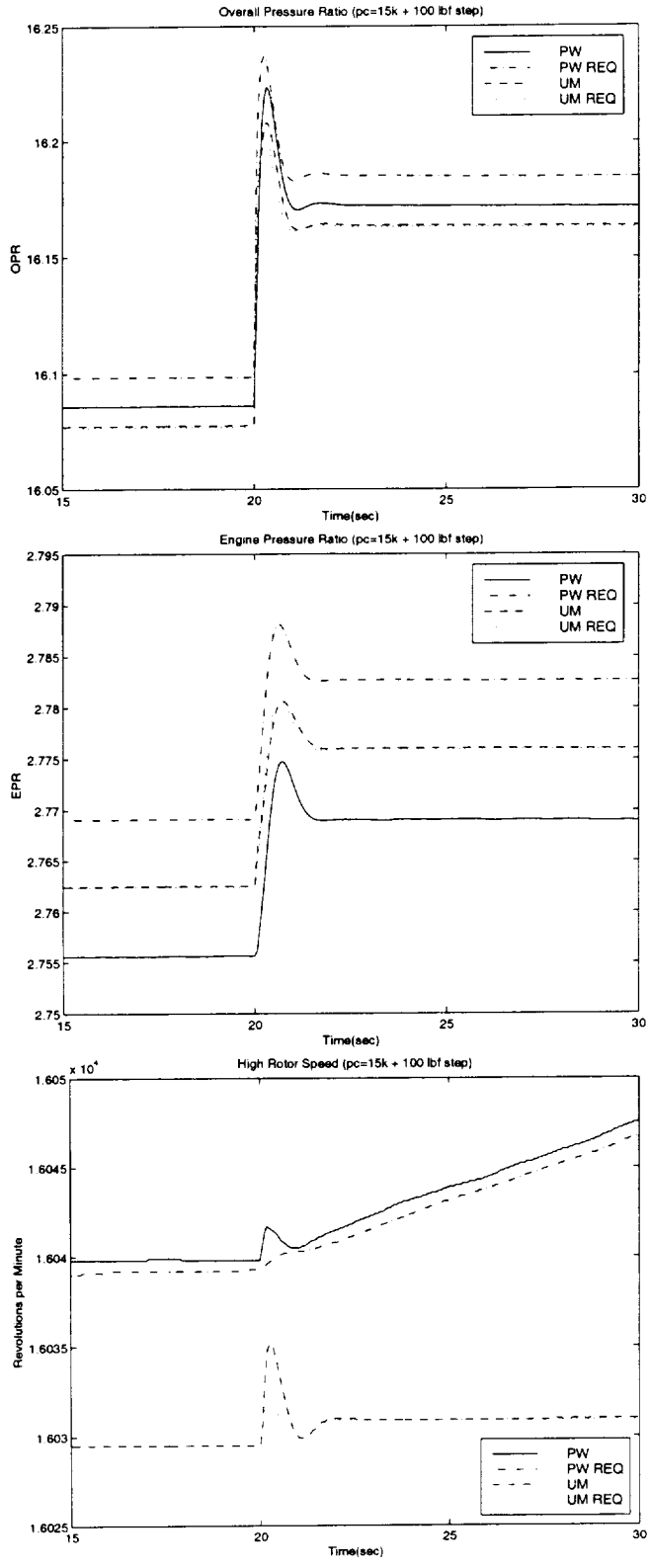

Figure 3.7: Controller Inputs of New $\mathcal{H}_\infty$ Controlled System

Chapter 4

Linear Parameter-Varying Control Theory

We begin with a brief introduction to gain scheduling based on linear parameter-varying representations. For a compact subset $\mathcal{P} \subset \mathcal{R}^s$, the *parameter variation set* $\mathcal{F}_\mathcal{P}$ denotes the set of all piecewise continuous functions mapping $\mathcal{R}$ (time) into $\mathcal{P}$ with a finite number of discontinuities in any interval. A compact set $\mathcal{P} \subset \mathcal{R}^s$, along with continuous functions $A : \mathcal{R}^s \to \mathcal{R}^{n\times n}$, $B : \mathcal{R}^s \to \mathcal{R}^{n\times n_d}$, $C : \mathcal{R}^s \to \mathcal{R}^{n_e\times n}$ and $D : \mathcal{R}^s \to \mathcal{R}^{n_e\times n_d}$ represent an nth order *linear parametrically varying* (LPV) system, whose dynamics evolve as

$$\begin{bmatrix} \dot{x}(t) \\ e(t) \end{bmatrix} = \begin{bmatrix} A(\rho(t)) & B(\rho(t)) \\ C(\rho(t)) & D(\rho(t)) \end{bmatrix} \begin{bmatrix} x(t) \\ d(t) \end{bmatrix} \tag{4.1}$$

where $\rho \in \mathcal{F}_\mathcal{P}$. For here on the time dependence of the parameter ρ will be suppressed due to space limitations. The induced $\mathcal{L}_2$ norm of a quadratically stable LPV system $G_{\mathcal{F}_\mathcal{P}}$ [5, 4, 7], with zero initial conditions, is defined as

$$\|G_{\mathcal{F}_\mathcal{P}}\| \doteq \sup_{\rho\in\mathcal{F}_\mathcal{P}} \sup_{\substack{\|d\|_2 \neq 0 \\ d \in L_2}} \frac{\|e\|_2}{\|d\|_2} \tag{4.2}$$

This quantity is always finite. The quadratic LPV γ-performance with bounds on the parameter rate-of-variation problem can now be stated.

Consider a generalized plant with the usual structure

$$\begin{bmatrix} \dot{x} \\ e_1 \\ e_2 \\ y \end{bmatrix} = \begin{bmatrix} A & B_{11} & B_{12} & B_2 \\ C_{11} & 0 & 0 & 0 \\ C_{12} & 0 & 0 & I \\ C_2 & 0 & I & 0 \end{bmatrix} \begin{bmatrix} x \\ d_1 \\ d_2 \\ u \end{bmatrix},$$

where the A, B, and C matrices are a function of ρ. For simplicity of derivation, assume $D_{11}(\rho) = 0, D_{22}(\rho) = 0$ and $D_{12}(\rho), D_{21}(\rho)$ have been scaled to the standard form. The LPV rate bounded synthesis solution can be solved as a two step procedure [5, 7],

1. Find NECESSARY AND SUFFICIENT conditions for existence of a dynamic controller of the form

$$\begin{bmatrix} \dot{x}_k \\ u \end{bmatrix} = \begin{bmatrix} A_K(\rho, \dot{\rho}) & B_K(\rho, \dot{\rho}) \\ C_K(\rho, \dot{\rho}) & D_K(\rho, \dot{\rho}) \end{bmatrix} \begin{bmatrix} x_k \\ y \end{bmatrix}.$$

   so that the closed-loop system passes the analysis test.

2. In the case of one parameter, eliminate from the realization of the controller the $\dot{\rho}$ dependence.

The analysis test is

Definition 1 There exists a LPV controller such that the closed-loop system passes the analysis test if and only if there exist matrix functions $X(\cdot)$ and $Y(\cdot)$ such that for all $\rho \in \mathcal{P}$, $X(\rho), Y(\rho) > 0$, and

$$\begin{bmatrix} Y\hat{A}^T + \hat{A}Y - \underline{\nu}(\rho)\frac{dY}{d\rho} - B_2B_2^T & YC_{11}^T & B_1 \\ C_{11}Y & -I_{n_{e1}} & 0 \\ B_1^T & 0 & -I_{n_d} \end{bmatrix} < 0$$

$$\begin{bmatrix} \tilde{A}^T X + X\tilde{A} + \underline{\nu}(\rho)\frac{dX}{d\rho} - C_2^T C_2 & XB_{11} & C_1^T \\ B_{11}^T X & -I_{n_{d1}} & 0 \\ C_1 & 0 & -I_{n_e} \end{bmatrix} < 0$$

$$\begin{bmatrix} X(\rho) & I_n \\ I_n & Y(\rho) \end{bmatrix} > 0$$

The matrices A, B, C, X and Y all depend on the parameter ρ. where

$$\begin{aligned} \hat{A}(\rho) &:= A(\rho) - B_2(\rho)C_{12}(\rho), \\ B_1(\rho) &= [B_{11}(\rho) \;\; B_{12}(\rho)], \\ \tilde{A}(\rho) &:= A(\rho) - B_{12}(\rho)C_2(\rho), \\ C_1^T(\rho) &= [C_{11}^T(\rho) \;\; C_{12}^T(\rho)]. \end{aligned}$$

The matrix functions $X(\cdot)$ and $Y(\cdot)$ can be solved for by expressing the above inequalities as the feasibility of a set of Affine Matrix Inequalities (AMIs), which can be solved numerically. For more details on LPV synthesis results the reader is referred to references [5, 6, 12]. Note that the parameter ρ is assumed to be available in real time, and hence it is possible to construct an LPV controller whose dynamics adjust according to variations in ρ and maintain stability and performance along all parameter trajectories.

This approach allows gain-scheduled controllers to be treated as a single entity, with the gain scheduling achieved via the parameter-dependent controller. This approach has been successfully applied to the synthesis of missile autopilots [6, 17] and flight controllers [8].

Chapter 5

LPV Control Design

LPV controllers were designed with the interconnection shown in Figure 5.1 to operate over the entire operating envelope. The same weights, sensor models and actuator models used for the second $\mathcal{H}_\infty$ design are used for the LPV controller. Here P_{sys} changes with the scheduling parameter ρ, which we chose as a lagged measurement of power code. Using LPV synthesis techniques guarantees both stability and performance in the presence of time variations of the time-varying parameter ρ. The technique requires solving a linear matrix inequality (LMI) over the entire parameter space.

The initial LPV designs in this chapter allow for infinity fast variations of the scheduled parameter ρ. These LPV controllers are called "non-rate bounded" designs. These may be inherently conservative since the controller must achieve the desired performance and robustness objectives for arbitrarily rapid changes in power code. Subsequent LPV design account for the physical limits on the rate-of-variation of power in the LPV synthesis process. These LPV controllers are called "rate-bounded" controllers. The rate-bounded LPV controllers are less conservative than the non-rate-bounded designs and more closely account for the physics of the physical system.

The control design objective was to synthesize a LPV controller which has the same decoupled command response across power code variation. The LPV controller measures the errors in OPR, EPR and N2 responses and schedules on power code. The resulting set of 10 controllers, one for each operating point, contained 20 states. An induced $\mathcal{L}_2$ norm of 2.3 was achieved.

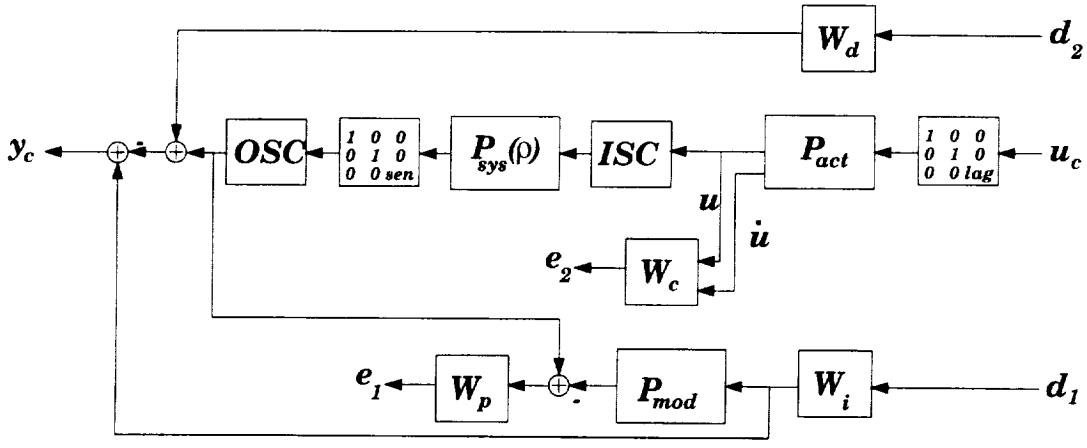

Figure 5.1: LPV Control Interconnection

5.1 LPV Control Algorithm

Integration of the LPV controllers into the ROCETS nonlinear simulation required a new controller Fortran subroutine. As with the $\mathcal{H}_\infty$ controller algorithm, we wanted to be able to handle controllers of various state order, without having to recompile the simulation. Lower order controllers were, as with the $\mathcal{H}_\infty$ algorithm, padded with zeros to have 21 states. The synthesized controllers were read in to the simulation from an external file. The controller entries for each point design were listed in a single column such that the 3000 lbf thrust power code entry A(1,1) was the first entry and the 30000 lbf thrust power code entry D(3,3) was the last entry. Figure 5.2 graphically displays the controller shape. The algorithm reads in the controller from the external file, assuming it contains controllers for ten operating points. Gains for all controllers not explicitly designed, are linearly interpolated from those of designed controllers. Figure 5.3 depicts an example of controller linear interpolation. Here controllers were explicitly designed for 6000 and 9000 lbf power code and 5000 and 6000 ft altitude. The gain for 8000 lbf, 5500 ft is easily interpolated. [1]

The trim values of the normalized actuator commands generated from the Jacobian linearizations are added to the LPV controller commands in the ROCKET engine simulation. These trim values are scheduled as a function of a lagged power code measurement and altitude. The addition of the trim value provided the steady state control input whereas the LPV control signals tracking the dynamic response of the system.

Fig. 5.4 shows 100 lbf step response results using a single LPV controller. Comparison with the $\mathcal{H}_\infty$ point designs step responses (Fig. 3.7) indicate that the LPV point controllers

[1] Chris Mitchel wrote much of the LPV interpolation code

have similar characteristics to the $\mathcal{H}_\infty$ counterparts. Hence we have recovered the linear performance and robustness of the original $\mathcal{H}_\infty$ point design with the LPV gain-scheduled design while directly synthesizing a globally stable, gain-scheduled multivariable controller. In contrast, however, the performance of the LPV controller is guaranteed not only at the design points, but at intermediate values of power code as well. Moreover, implementation of the gain–scheduled LPV controller requires only linear interpolation of the state–space data.

The LPV controller was simulated in ROCETS with power code variations the operating envelope (from 3000 to 27000 lbf thrust) at sea level, standard atmosphere and zero Mach. Figure 5.5 shows the commanded power code variation as a function of time. Figure 5.6 shows the plant outputs for the LPV controller and the baseline controller along with the requests for each. Figure 5.7 shows the plant inputs. The zoomed area in each figure provides a closer comparison of the LPV and baseline controllers performance. The plots indicate the LPV controllers reach compatible tracking, and response as the baseline controllers. This affirms the LPV controller synthesis methodology can achieve the same results as the more time-consuming ad-hoc methods traditionally used.

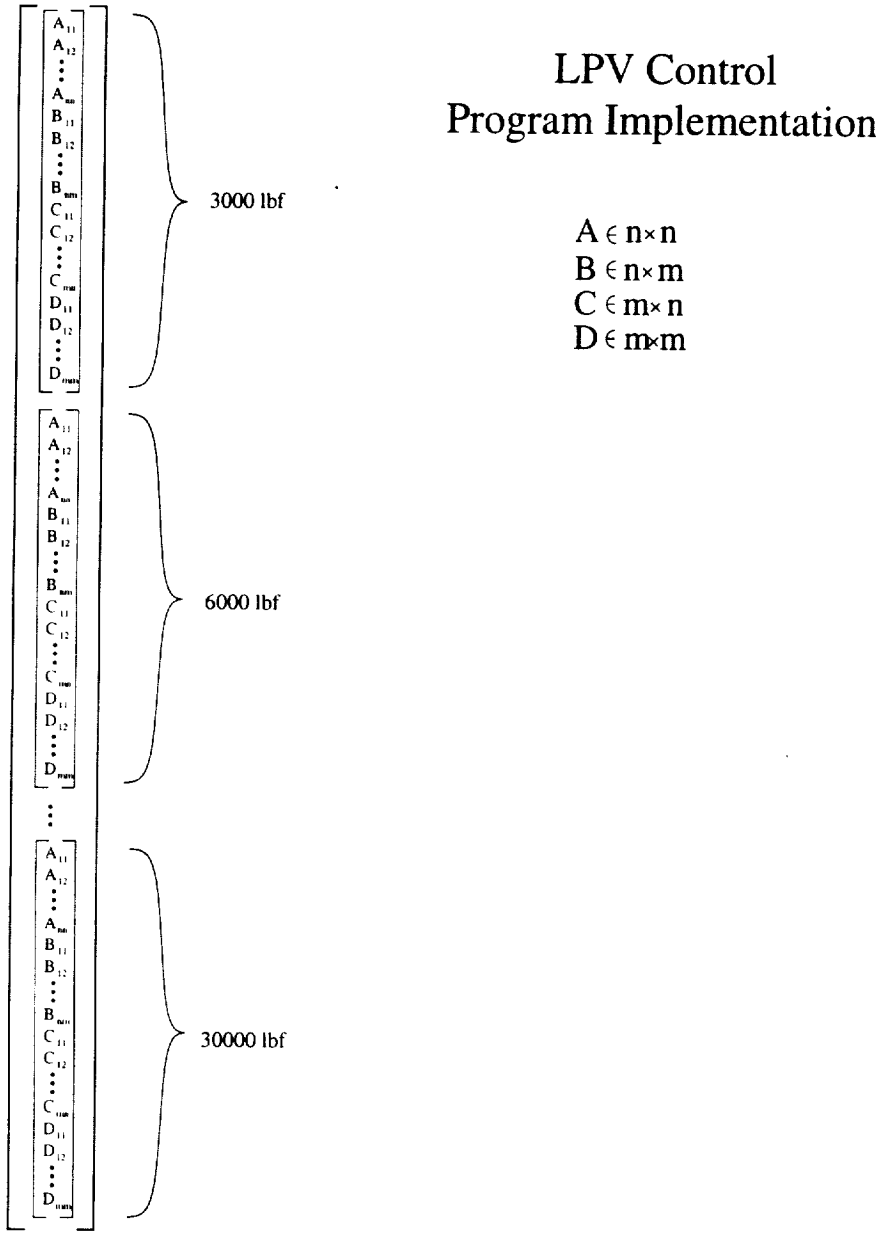

Figure 5.2: LPV Control Implementation Framework

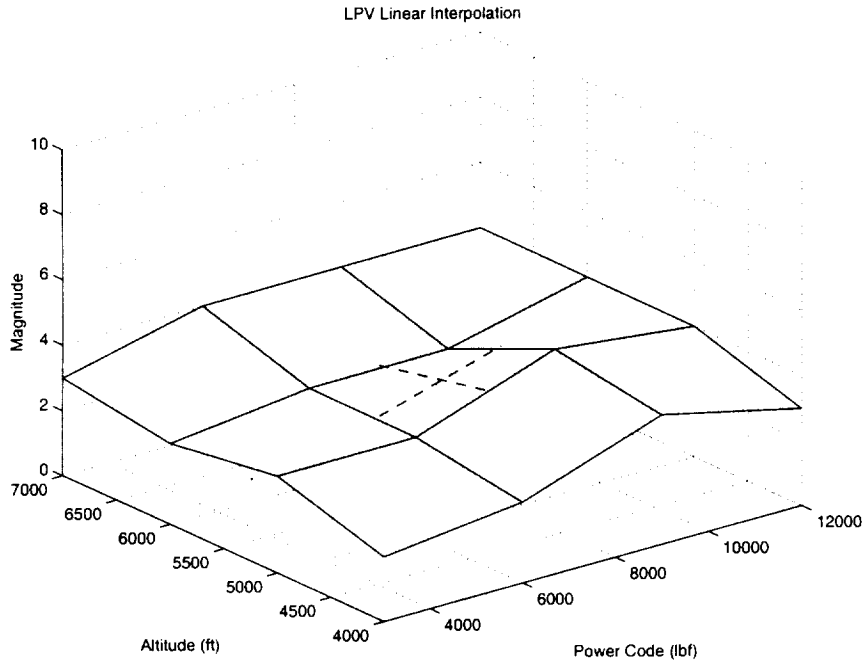

Figure 5.3: LPV Control Linear Interpolation

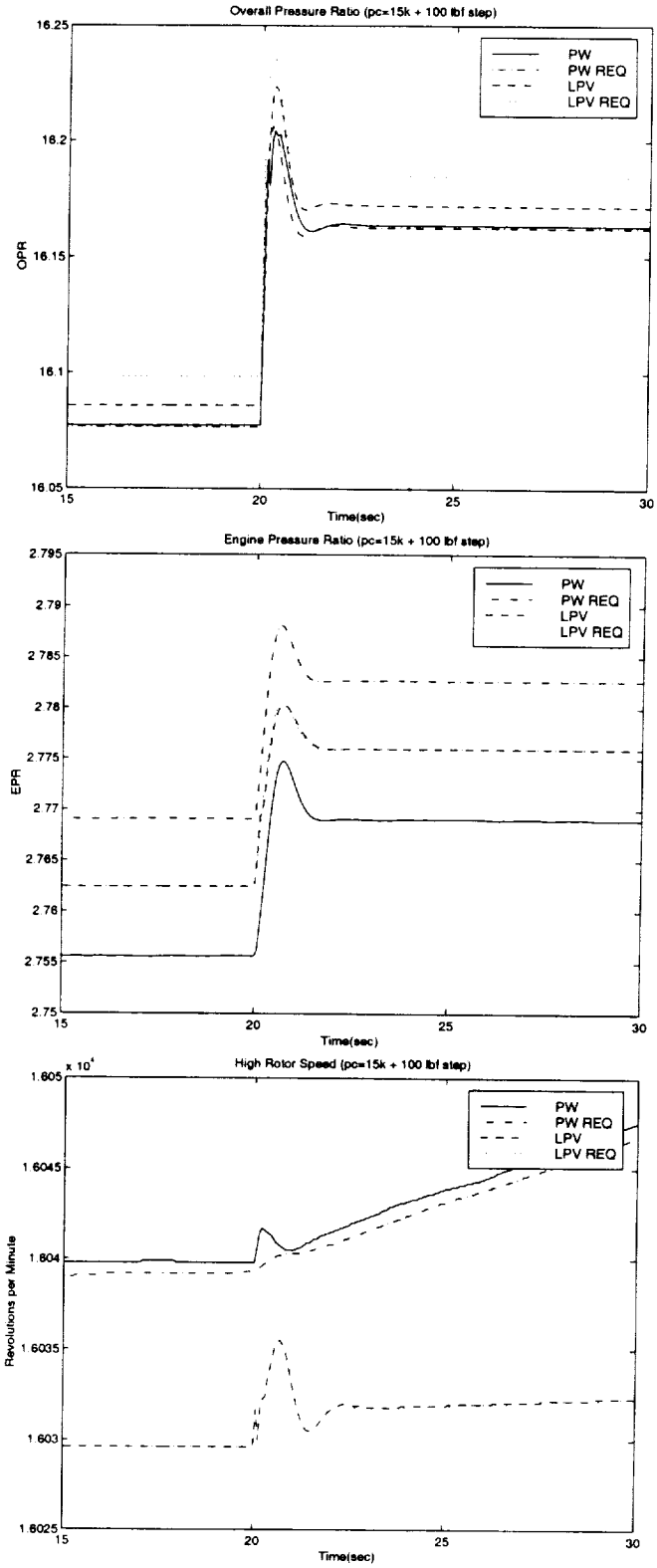

Figure 5.4: LPV 100 lbf Step Response

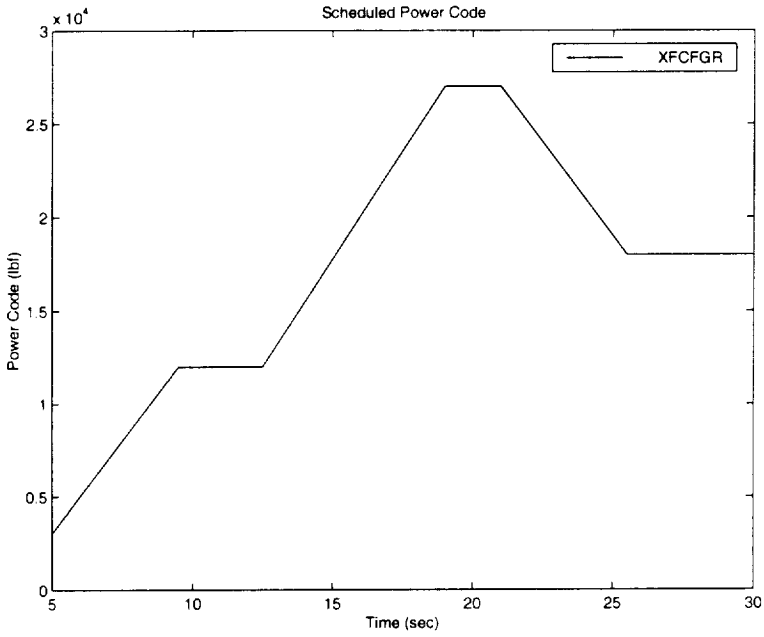

Figure 5.5: Demanded Simulation Power Code

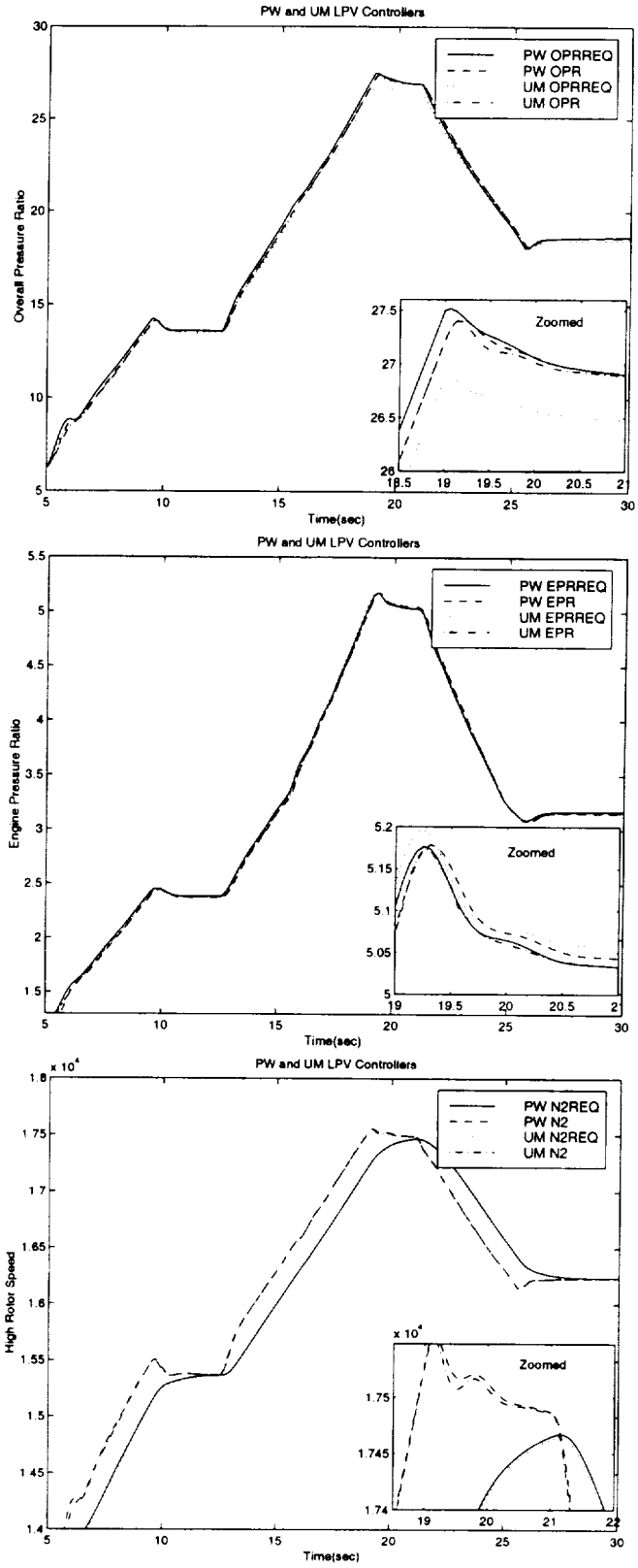

Figure 5.6: LPV and baseline Controller Plant Outputs

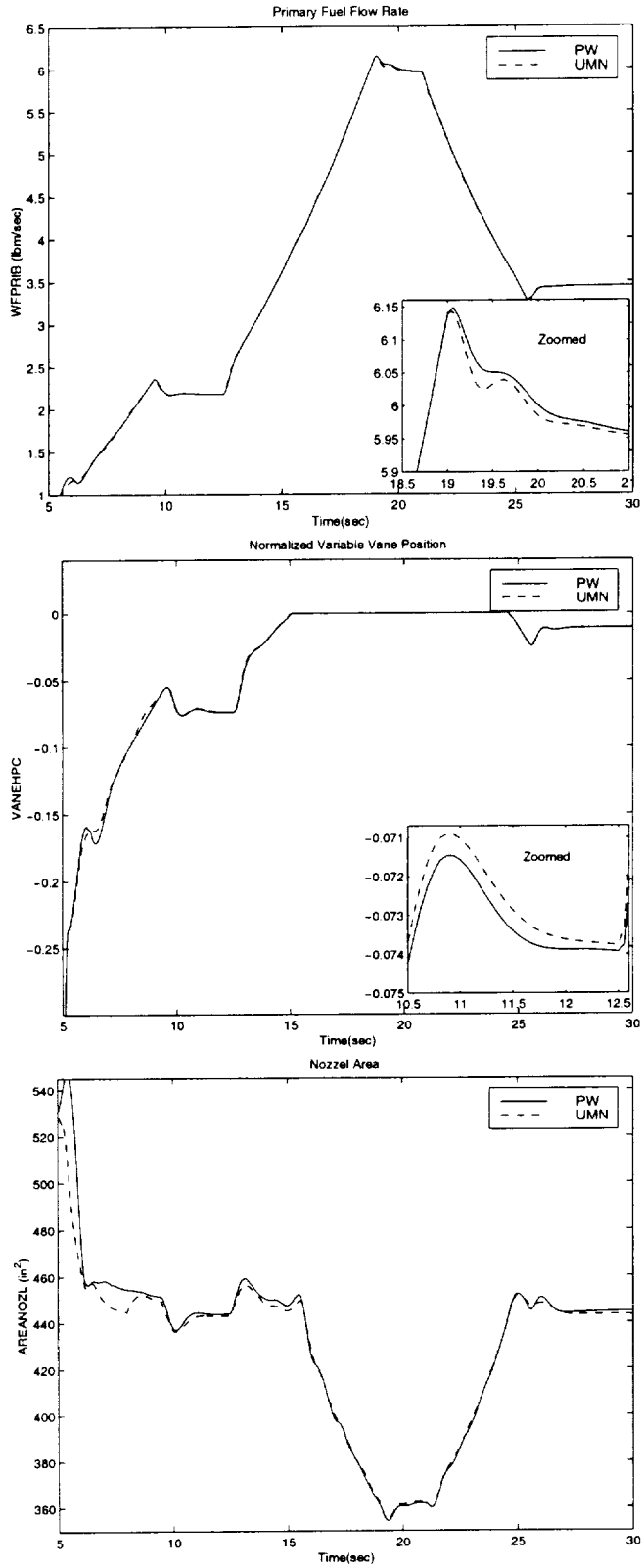

Figure 5.7: LPV and baseline Controller Plant Inputs

Chapter 6

LPV Controller Redesign

This chapter describes the redesign of the LPV controller. The LPV controller described in the previous chapter did not perform well at high altitudes. Hence the goal of the controller redesign is to synthesize a single LPV controller that performs well across the flight envelope for all environmental conditions. The environmental conditions include a polar day (-18.5°C), standard day (24°C) and a tropical day (41°C). The objective is also to schedule only on lagged power code. Noted that above 30K ft, the actuators begin to saturate. The effect of these saturations will be investigated since they directly effect the achievable performance of the closed-loop system and potentially the stability of the system.

$\mathcal{H}_\infty$ and LPV control theory are used to synthesize feedback controllers. The system interconnection initially considered for control design is shown in Fig. 6.2. In the figure, $\hat{P}(\rho)$ represents the three-input, three-output, three-state engine model with the scaling normalizing matrices (ISC and OSC) and sensor model P_{sen}. P_{sen} is defined as

$$P_{\text{sen}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & \frac{\frac{1}{20}s+1}{\frac{1}{4}s+1} \end{bmatrix}$$

A block diagram picture of $\hat{P}(\rho)$ is shown in Fig. 6.1.

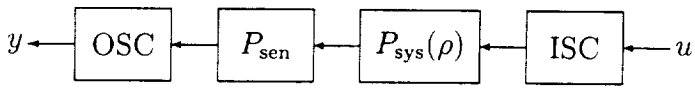

Figure 6.1: Turbofan engine plant model $\hat{P}(\rho)$

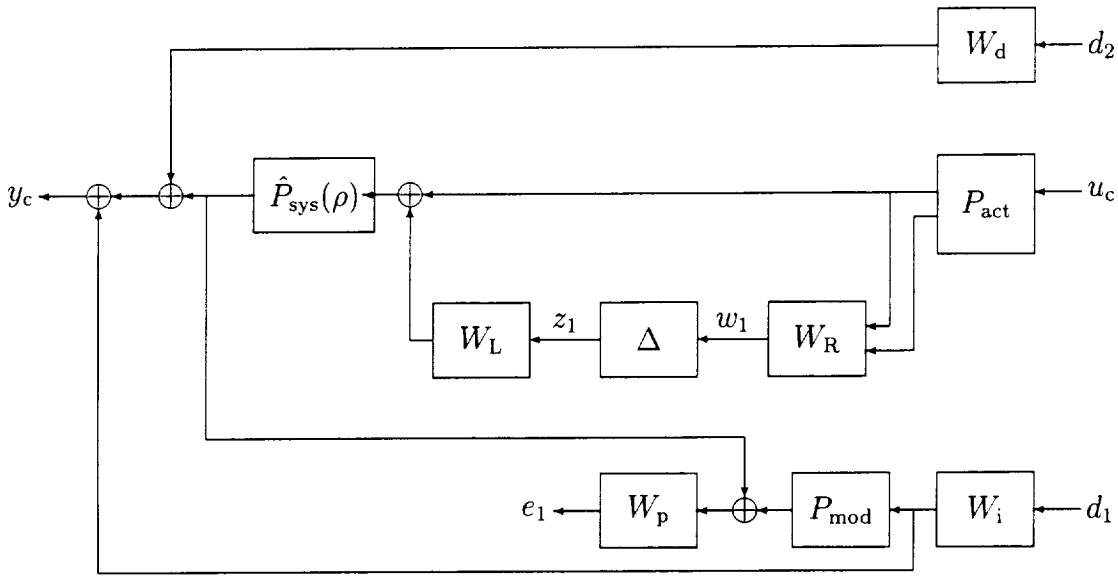

Figure 6.2: Interconnection for LPV controller redesign.

The A, B, C, D matrices vary as a function of power code (PC), with data provided for ten distinct power codes $PC = 3000, 6000, \ldots, 30000$. The input and output units have been normalized. As mentioned previously, inputs to the plant are primary burner fuel flow rate (WFPRIB or PBFF, lbm/sec), high compressor vane percentage area (VANEHPC or CVA) and convergent throat area feedback (AREANOZL or CVA, in^2). Outputs are normalized overall pressure ratio (OPR), engine pressure ratio (EPR), and high rotor speed (N2). The PW turbofan models correspond to the identified linear model described in Section 3.1.

The control problem is formulated as a model matching problem in the $\mathcal{H}_\infty$ and LPV framework. We desire the engine to respond as three single–input/single–output (SISO) systems with no off–diagonal coupling. P_{mod} represents such a decoupled system, and any difference between this desired model and the true plant is penalized via the weight W_p.

The disturbance vector d_1 represents customer demands, whose values we wish the plant output to track. Specifically these commands correspond to desired overall engine pressure ratio, engine pressure ratio and high rotor speed. The disturbance vector d_2 represents noise or modeling error, to which the controller should be robust. By keeping the error vector e_1 small, good tracking of d_1 is ensured. Input and actuator modeling errors, actuator magnitude and rate constraints and closed-loop bandwidth constraints are accounted for in the control design via the z_1 to w_1 input/output pair. The input measurements to the controller is y_c and the controller generates u_c as control commands.

The actuator model P_{act} was derived from first principles model of the actuators in the

ROCETS engine simulation and system identification techniques (see Section 3.1). The modeling objective was to match the time response of the ROCETS engine simulation with the linearized actuator, sensor and engine models are power code equilibrium points. The actuator and sensor models were derived for the sea level, standard atmosphere and zero velocity flight condition. The actuator model is

$$P_{act} = \begin{bmatrix} \frac{20}{s+20} & 0 & 0 \\ 0 & \frac{20}{s+20} & 0 \\ 0 & 0 & \frac{5}{s+5} \\ \frac{20s}{s+20} & 0 & 0 \\ 0 & \frac{20s}{s+20} & 0 \\ 0 & 0 & \frac{5s}{s+5} \end{bmatrix}.$$

Its realization has three states. The actuator model depicted in the Figure 6.2 outputs actuator positions u and actuator rates $\dot{u}$, to allow actuator rates to be penalized in the control design. Note that actuator associated with the convergent throat area nozzle (AREANOZL) is represented differently than the actuator model identified in Section 3.1. To minimize the number of states in the control design problem, the original first order actuator model with a lag is replaced by a first order transfer function model that captures the phase characteristics of the original two state model.

The model-matching block P_{mod} was based on previous work in Reference [2]. The engine variables OPR and N2 are dynamically coupled based on the physics of the turbofan engine. Therefore the controller is designed to have the response of the engine high rotor speed (N2) lag the response of OPR and EPR. It is desired that the engine response to OPR and EPR request follow a second order model with natural frequency of 12 rad/sec and damping of 0.85. The engine response to N2 request should follow a second order model with natural frequency of 2.5 rad/sec and damping of 0.85. The original engine model contained scaling matrices ICS and OSC whose role was to normalize the input and output signals to controller relative to one another. The input weight W_i is selected to be the $I_{3\times3}$ matrix. Therefore, the normalized EPR, OPR and N2 commands are modeled as being of the same relative magnitude.

Within the $\mathcal{H}_\infty$ framework, the error between the desired engine response and the actual engine response are weighted in magnitude and across frequency via the weight W_p. The ideal models were derived based on the desired low frequency response of the engine. From a performance perspective, matching the ideal model response is more important at low frequency than high frequency (above 20 rad/s). Hence the performance weight W_p, which penalizes the difference between the desired and the actual closed–loop response of the turbofan engine, reflects the low frequency tracking accuracy desired. The larger the magnitude of W_p, the smaller the allowable difference between desired and actual output. In our designs,

W_p is a first order low–pass weight given by

$$W_p = \begin{bmatrix} 200\frac{\frac{s}{1500}+1}{\frac{s}{0.015}+1} & 0 & 0 \\ 0 & 160\frac{\frac{s}{1500}+1}{\frac{s}{0.2}+1} & 0 \\ 0 & 0 & 160\frac{\frac{s}{500}+1}{\frac{s}{0.25}+1} \end{bmatrix}$$

This ensures good tracking of the response models in the bandwidth 1–10 rad/s, with little or no DC error. At low frequency (below 0.1 rad/s) the W_p weight on the OPR corresponds to DC errors of $\frac{1}{200}$ or 0.5% tracking error. The EPR and N2 tracking error at DC can be $\frac{1}{160}$ or 0.625%. At high frequency, the mismatch between actual and desired output is not penalized. A magnitude plot of W_p is shown in Figure 6.3.

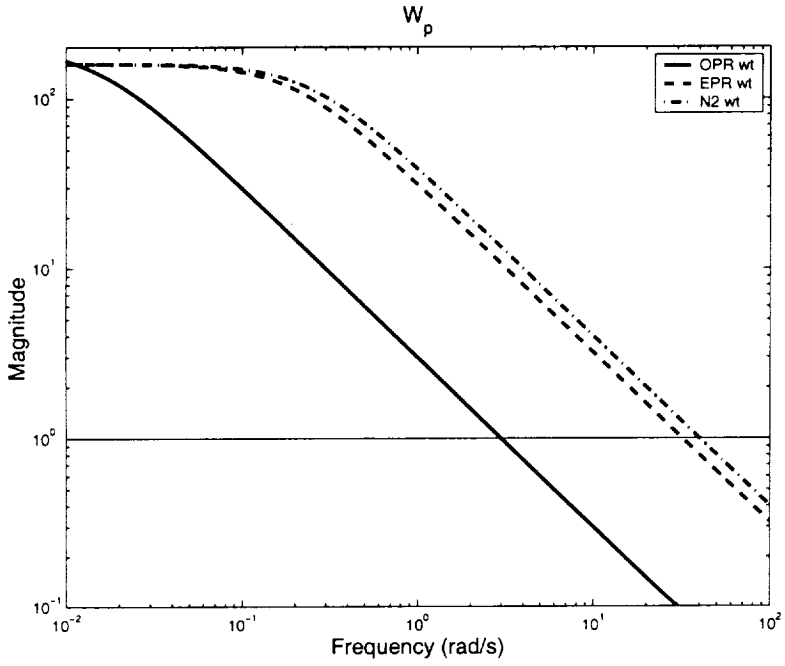

Figure 6.3: Tracking Performance Weight W_p

An output disturbance model is included in the controller synthesis problem formulation to provide robustness of the closed-loop system. The disturbance weight W_d is used to account for errors between the linear engine model and the nonlinear model as well as limit the bandwidth of control effort by ramping up at high frequency. Placing the disturbance at the input to the engine model would require the weight W_d to possibly vary with power code, as the plant gains change significantly across power code. To simplify the LPV gain–scheduled control design, the disturbance model is put at the plant output, where a first–order weight was found to be sufficient to limit controller bandwidth. We shall denote the output

disturbance weight by W_d.

$$W_d = 8 \times 10^{-5} \frac{\frac{s}{0.03} + 1}{\frac{s}{30000} + 1} I_{3\times 3}$$

Multiplicative input uncertainty is included in the design to provide robustness to actuator and input uncertainty, limit the control magnitude and rate commands and limit the bandwidth of the closed-loop system. The uncertainty, modeled by Δ in Figure 6.2, is weighted on the left by W_{L} and on the right by W_{R} to balance its effect on the $\mathcal{H}_\infty$ control design. W_{L} is a diagonal constant matrix, $1.76 I_{3\times 3}$ and W_{R} is a constant 3×6 matrix

$$W_{\mathrm{R}} = \left[\begin{array}{cc} 0.9 I_{3\times 3} & \left[\begin{array}{cc} 0.23 I_{2\times 2} & 0 \\ 0 & 0.28 \end{array} \right] \end{array} \right].$$

The actuator deflections and rates are input to W_{R}. They are used to generate a frequency dependent uncertainty weight without the introduction of additional states to the design. Figure 6.4 shows the frequency response of the three input uncertainty weights. These weights imply that there is 16% uncertainty at low frequency in each input channel. At 2.5 rad/s, the uncertainty in the first two channels has risen to 100% whereas channel three reaches 100% input uncertainty at 3.2 rad/s. The amount of model uncertainty is significant and does interact with the ability of the control design to meet the performance specifications. Since the uncertainty model and performance specifications are in conflict, i.e. performance is desired at frequencies for which the phase of the engine model is complete unknown (100% or more input uncertainty), the $\mathcal{H}_\infty$ controller will not be able to achieve an infinity norm less than 1. The lowest achievable $\mathcal{H}_\infty$ norm for the system for the given uncertainty weights W_{R} and W_{L} and performance weight W_{p} is 5.

$\mathcal{H}_\infty$ controllers are synthesized for the open-loop intereconnection shown in Figure 6.2.

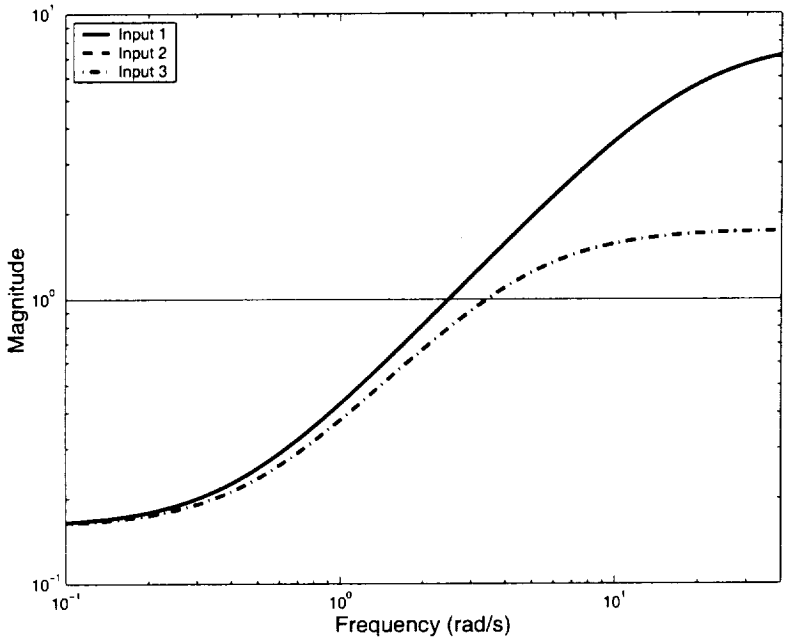

Figure 6.4: Input Multiplicative Uncertainty Weights

6.1 $\mathcal{H}_\infty$ Linear Point Designs

Ten $\mathcal{H}_\infty$ controllers were synthesized, one for each plant, using the interconnection of Fig. 6.2. The $\mathcal{H}_\infty$-norm achieved for these designs ranged between 5.5 and 7.9. The 7.9 $\mathcal{H}_\infty$-norm is associated with the 30000 power code and the 5.5 norm is associated with the 12000 power code. As the interconnection used has 19 states, so do each of the ten controllers. In this study, no attempt is made to reduce the order of the $\mathcal{H}_\infty$ linear controllers.

Performance objectives were evaluated in the frequency and time domain. The frequency analysis corresponds to the H_∞-norm from d to e, which we desire to be smaller than 1. As noted in the previous section, there is a conflict in the level of modeling input uncertainty in the frequency range of 1 to 20 rad/s and the performance objectives. The performance and robustness requirements could not be achieved based on the problem formulation. Hence, H_∞-norm values of as large as 8 were considered acceptable. The second judgment was made in terms of our specified objective, through inspection of step responses. Tracking error, decoupling, magnitude of actuator positions and rates were all evaluated in terms of our original goals using the nonlinear ROCETS simulation of the turbofan engine.

The step response simulations for the linearized engine model with $\mathcal{H}_\infty$ point design are shown in Fig. 6.5. Fig. 6.5 demonstrates the ability of individual controllers (one designed for each power code) to track the step commands. There are nine top plots in Fig. 6.5 corresponding to the three command inputs: $\mathrm{OPR_{ref}}$, $\mathrm{EPR_{ref}}$, and $\mathrm{N2_{ref}}$ to outputs of the engine model: OPR, EPR, and N2. The control design objective was to have a completely decoupled response from the reference commands to the engine outputs. Note that the dynamics of the engine vary significantly with power code. Based on the results of the linear, $\mathcal{H}_\infty$ point design controllers some inherent coupling exists between the diagonal and off-diagonal.

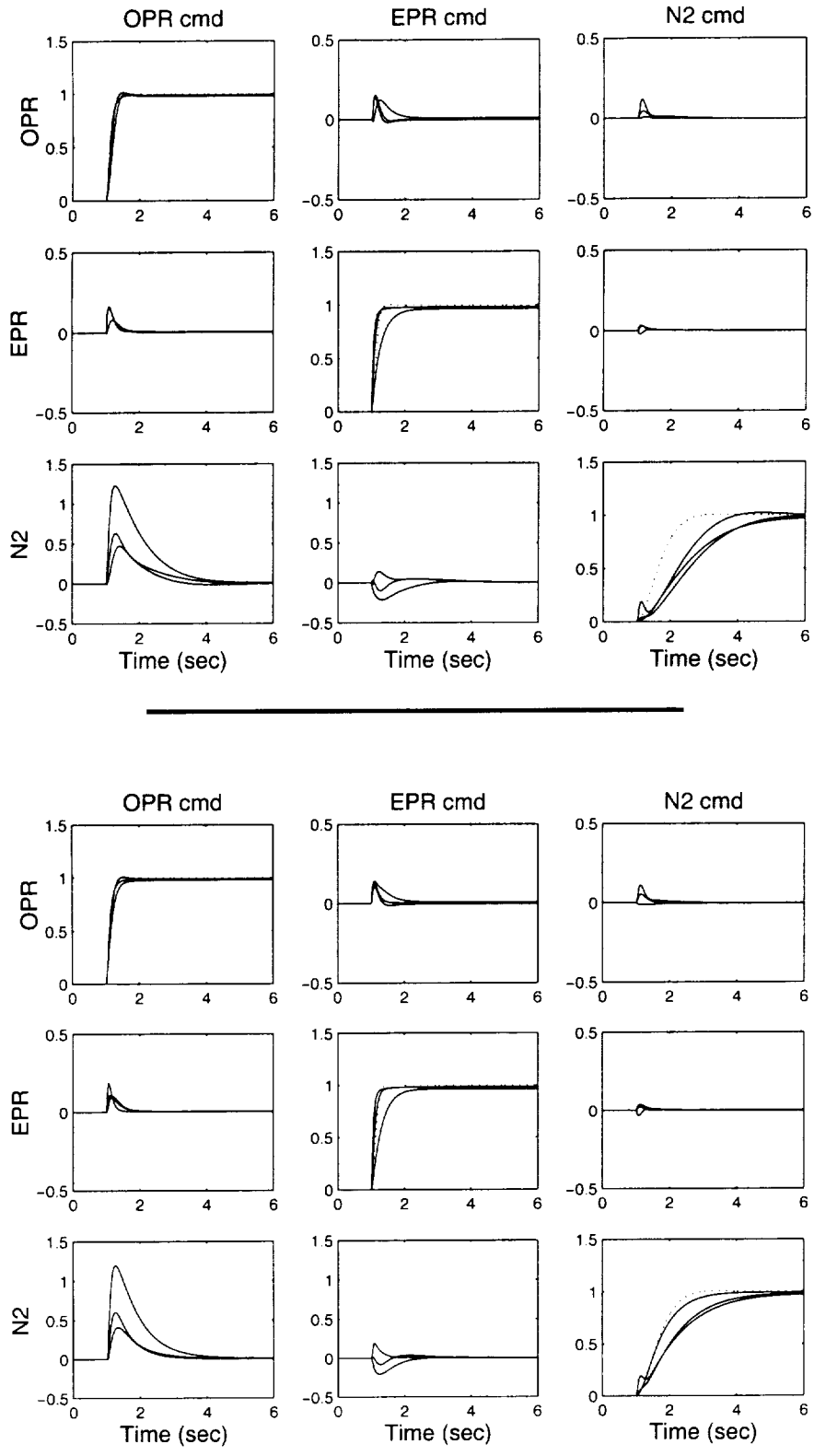

Figure 6.5: Unit step response at power code 3K, 15K, 30K: $\mathcal{H}_\infty$ Point Designs (top), LPV (bottom)

For brevity, we will discuss the robustness analysis of three point designs at 3000, 15000 and 30000 power codes. A plot of the magnitudes of the $\mathcal{H}_\infty$ point design controllers is shown in Figure 6.6. Based on this plot, observe that the magnitude of the $\mathcal{H}_\infty$ point designs change across power code. This is consistent with the behavior of the open-loop engine model Jacobian linearizations. Loop-at-a-time gain and phase margins were calculated for all the $\mathcal{H}_\infty$ point design controllers. They achieved at least 27 dB of gain margin at a frequency of 0.5 rad/sec and 74 degrees of phase margin at a 0.5 rad/sec. Multivariable input and output sensitivity and complementary sensitivity plots for the $\mathcal{H}_\infty$ controllers (solid lines) were calculated for the three power codes, see Figure 6.7. The input sensitivity and complementary sensitivity plots for the $\mathcal{H}_\infty$ point design are excellent with peaks below 1.3 across frequency. The output sensitivity and complementary sensitivity plots are very good at the low and mid power codes with peaks less than 2.2, though they degrade to a peak of 2.7 at 3 rad/s for the 30000 power code. This corresponds to the closed-loop system being robustness to 35% full block uncertainty at the output of the system, If this is deemed unacceptable, the weighting functions in the initial problem formulation may need to be modified to improve the output sensitivity and complementary sensitivity of the $\mathcal{H}_\infty$ controllers at high power codes.

Hence the closed-loop system is more sensitive to modeling errors at the output of the plant than the plant input. This is to be expected since the control problem formulation indicated that there was significantly more modeling error and uncertainty at the input to the plant than at the plant output. The point design controller are not meant to be scheduled across the operating envelope, rather the $\mathcal{H}_\infty$ point design controller form a baseline for comparison with the LPV control designs. The LPV controller is designed for the entire operating envelope and is implemented in a nonlinear ROCETS simulation of the engine for testing.

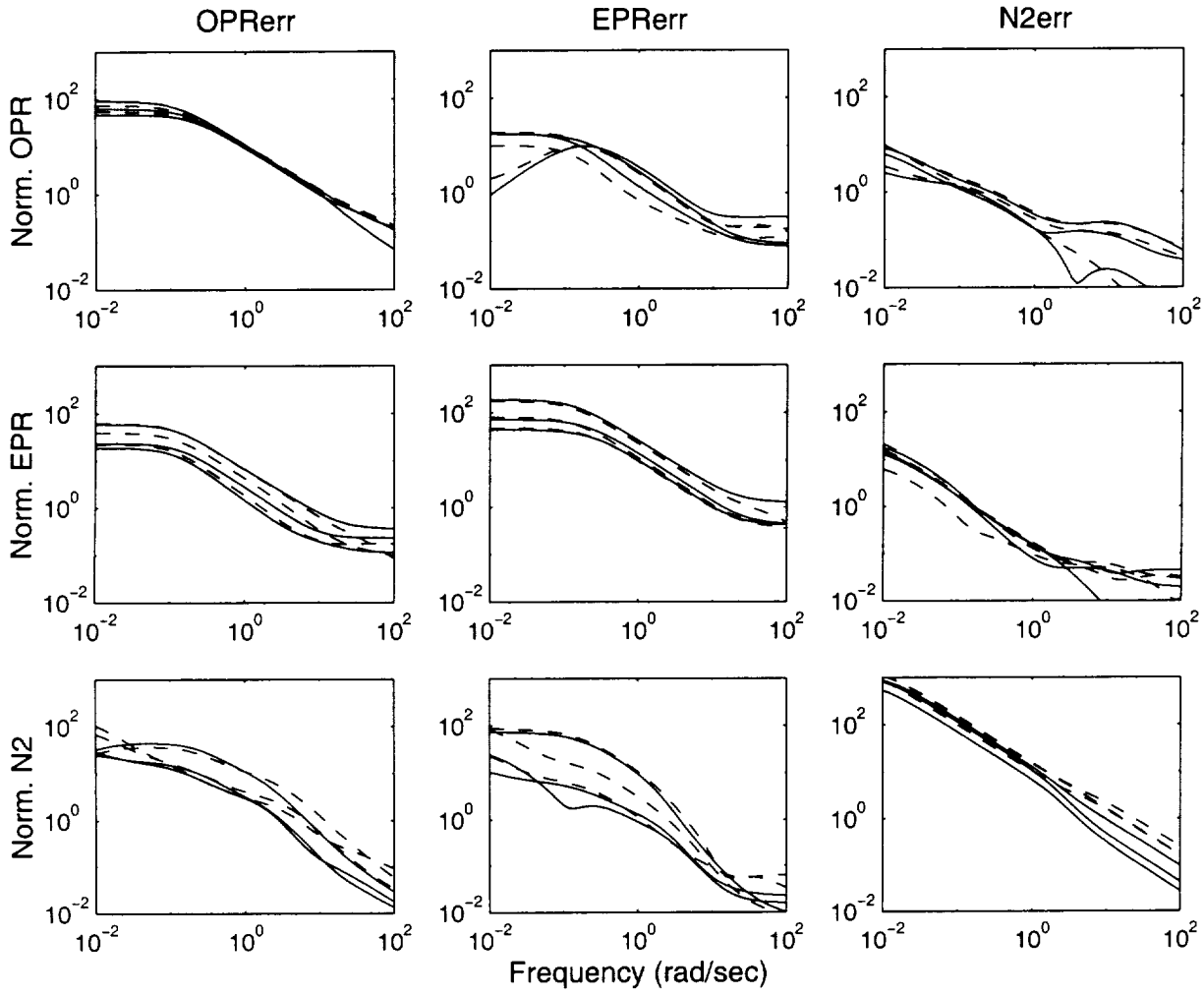

Figure 6.6: Magnitude frequency response at power code 3K, 15K, 30K: $\mathcal{H}_\infty$ (solid), LPV (dashed)

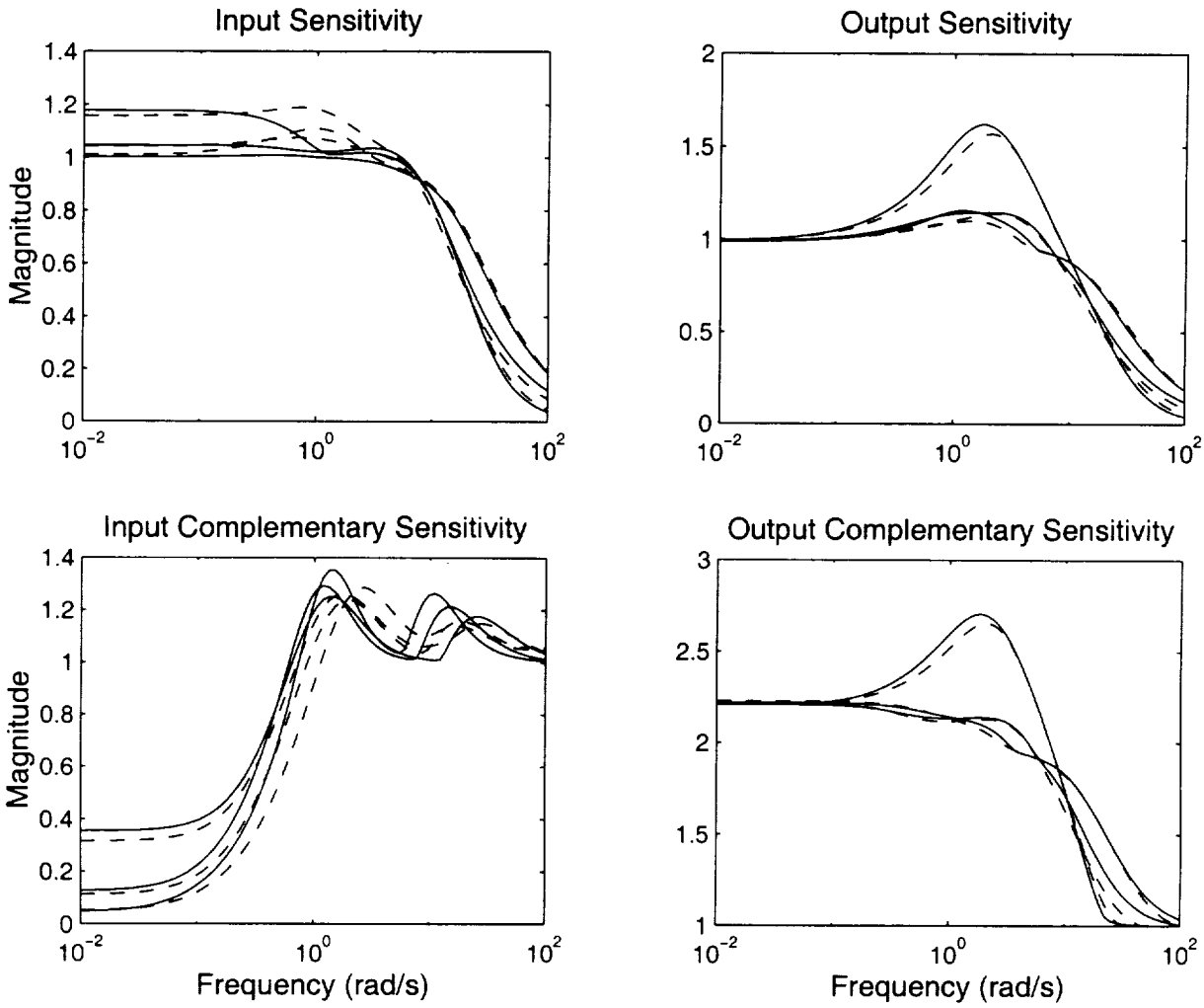

Figure 6.7: Sensitivity/complementary sensitivity at 3K, 15K, 30K: $\mathcal{H}_\infty$ (solid), LPV (dashed)

6.2 LPV Design

The control design objective is to synthesize a controller which has decoupled command response across power code variation. The flight envelope considered is sea level to 40,000 ft with environmental conditions varying from a polar to tropical day. This LPV controller was synthesized using the interconnection in Fig. 6.2. The design model was based on ten Jacobian linearization models at sea level, zero velocity and standard atmosphere conditions. The same weighting functions used in the $\mathcal{H}_\infty$ point designs are used in the LPV design, though the turbofan engine model varies as a function of power code.

The LPV controller measures the errors in OPR, EPR and N2 responses and schedules on a lagged measurement of power code. Recall that the commands to OPR, EPR and N2 are generated by an outer-loop algorithms that is part of the PW STF 952 ROCETS simulation. The outer–loop commands are a function of power code, flight condition, environmental conditions and current state of the engine. The focus of this study is tracking these commands since we can not directly effect them. The LPV synthesis algorithms require solving a linear matrix inequality (LMI) over the entire parameter space. This is an infinite dimensional problem. A finite dimensional LMI problem is derived by considering ten points of the flight envelope. Recall from the $\mathcal{H}_\infty$ point designs that the $\mathcal{H}_\infty$-norm varied from 5.5 to 7.9 across power code. To equalize the objectives across operating points, we have found it beneficial to normalize each point model by the inverse of its achieved linear, time-invariant H_∞-norm. After this scaling, each linear point design would now achieve an H_∞-norm of 1. Hence, the induced $\mathcal{L}_2$ norm generated by the LPV controllers can be compared with the $\mathcal{H}_\infty$ point designs.

All LPV controllers were synthesized using 3-input/3-output state-space models of the turbofan engine starting at 3K power code (idle) up to 30K power code, military power, with a model every 3K. The Jacobian linearized models are used in the LPV design. A lookup table is constructed for the steady-state control input values as a function of power code and altitude. These steady-state trim inputs are added to the LPV control inputs to generate the actual command to the engine. The LPV controllers are synthesized using the μ-Analysis and Synthesis and LMI Matlab Toolbox algorithms on a Red Hat Linux computer with a 866 Mhz PIII processor. Both the non-rate bounded and rate bounded design include the time taken to limit high frequency poles of the controller.

The non-rate bounded control design makes no assumption on the rate of variation of the scheduled parameter, lagged power code. In essence, it allows this parameter to vary infinitely fast. This may be overly conservative since the non-rate bounded LPV controller is required to satisfied the prescribed performance and robustness requirements simultaneously

at all power codes. As with the $\mathcal{H}_\infty$ point designs, the non-rate bounded LPV controller has 19 states. No attempt was made to reduce the state order of the non-rate bounded design.

The induced $\mathcal{L}_2$ norm for the non-rate bounded LPV controller was 3.53 and took 121 CPU seconds to synthesize. The value of the induced $\mathcal{L}_2$ norm corresponds to a factor of 3.53 degradation in the performance and robustness norms of the non-rate bounded controller relative to the $\mathcal{H}_\infty$ point design results.

The rate-bounded LPV control design directly accounts for the rate of variation of the scheduled parameter, lagged power code. The rate-bounded LPV design is an approximation of an infinite-dimensional problem as a finite-dimensional problem with a fixed set of basis functions. The solution to the rate-bounded controller equations are based on parameter dependent X and Y scalings. Three basis functions are considered to describe the X and Y LPV solution matrices: the constant 1, a function of power code and the square of power code. Denoting power code as ρ, the X and Y solutions matrices are:

$$\begin{aligned} X(\rho) &= X_0 + \rho X_1 + \rho^2 X_2 \\ Y(\rho) &= Y_0 + \rho Y_1 + \rho^2 Y_2 \end{aligned}$$

Currently there is no systematic approach to the selection of basis function. Our experience has lead us to select basis functions that correspond to physical parameters that directly effect the dynamic response of the plant model within the flight envelope. Selecting power code and the squared of power code relates to our observations of how the linearized dynamics of the turbofan engine change as a function of power code. The bound on the rate-of-variation of power code is taken to be ± 8000 /sec.

Two non-rate and rate bounded LPV controllers are synthesized for each control problem. The first formulates the standard non-rate/rate bounded LPV control algorithm. The second uses the first solution and includes an additional constraint on the closed-loop system poles at the grid points. The reason for the design of the second LPV controller is that the initial LPV design often has very high frequency controller poles at the grid points. Since only *ad hoc* model reduction algorithms are available, which may not eliminate the high frequency poles, a constraint on the closed-loop system poles is added to the original LPV design problem, the achievable induced $\mathcal{L}_2$ norm is relaxed 5% from its value in the initial design and the LMI problem is resolved. All the LPV controllers discussed are based on the second LMI solution with constraints on the closed-loop poles incorporated into the LMI equations.

For the rate bounded LPV control design, the induced $\mathcal{L}_2$ norm for all parameter trajectories was 1.16. This indicates that the performance/robustness levels can be up to 1.16 times worse that the corresponding linear point designs. A comparison between the induced $\mathcal{L}_2$ norm of the rate-bounded and non rate-bounded design indicates that the rate-bound nearly recovers the performance of the $\mathcal{H}_\infty$ point design (normalized to a norm of 1) with a factor

of 3 reduction in the non rate-bounded induced norm. The rate-bounded LPV design with three basis functions took 3501 CPU seconds to synthesize, a factor of 30 longer than the non-rate bounded design. The extra computation is due to the increased number of basis function parameters solved in the LMI optimization.

As with the point designs, the LPV controller has 19 states. A model reduction approach, based on the alignment of the LPV grid point controller eigenvectors was applied to the rate-bounded LPV controller. The rate-bounded controller was reduced to 12 states with no degradation in the induced norm or performance/robustness and could be reduced to 7 states with little degradation. The original 19 state rate-bounded design had high frequency poles at the grid points on the order of 2,000 rad/sec. The reduced 12 state controller had its high frequency poles below 200 rad/sec and the 7 state controller had high frequency poles below 50 rad/sec. All analysis and simulation results presented for the rate-bounded controller are based on the 12 state reduced order rate-bounded LPV design.

Fig. 6.5 (bottom figure) shows step response results using a the grid point LPV controllers at the equilibrium points. There is significant decoupling between the channels, though not quite as good as that of individual point designs. (The point designs represent the best we could hope to do for frozen parameter values). The poorest response in Fig. 6.5 is associated with the LPV controller response at the 3000 power code. In contrast, however, the performance of the LPV controller is guaranteed not only at the design points, but at intermediate values of power code as well. The implementation of the gain–scheduled LPV controller requires only linear interpolation of the state–space data for implementation.

Consider the robustness analysis of the LPV evaluated at power codes 3000, 15000 and 30000. A plot of the magnitudes of the rate-bounded LPV controller is shown in Figure 6.6. The magnitude of the gain-scheduled LPV design at the grid points is very similar to the optimal $\mathcal{H}_\infty$ point designs. Loop-at-a-time gain and phase margins were calculated and all the LPV point controllers achieve at least 27 dB of gain margin at a frequency of 3 rad/sec and 72 degrees of phase margin at a 1.9 rad/sec. Multivariable input and output sensitivity and complementary sensitivity plots for the $\mathcal{H}_\infty$ controllers (dashed lines) were calculated for the three power codes, see Figure 6.7. The input sensitivity and complementary sensitivity plots for the LPV grid point designs are excellent with peaks below 1.3 across frequency. The output sensitivity and complementary sensitivity plots are very good at the low and mid power codes with peaks less than 2.2, though they degrade to a peak of 2.7 at 3 rad/s for the 30000 power code. This corresponds to the closed-loop system being robustness to 35% full block uncertainty at the output of the system. It is obvious from the loop-at-a-time gain and phase margins and Figure 6.7 that the rate-bounded LPV design recovers all the robustness properties of the $\mathcal{H}_\infty$ point design controllers which is what was desired.

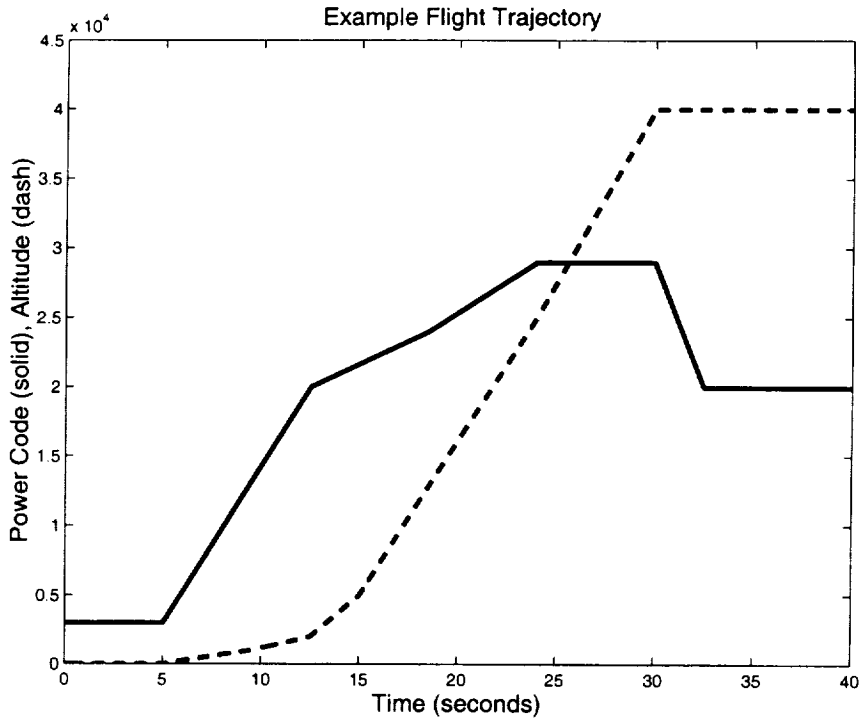

Figure 6.8: Power code (solid) and altitude (dashed) as a function of time

Comparison with the $\mathcal{H}_\infty$ point designs, linear step responses of the LPV point controllers indicate that the LPV point controllers have similar characteristics to the $\mathcal{H}_\infty$ counterparts. Hence we have recovered the linear performance and robustness of the original $\mathcal{H}_\infty$ point design with the LPV gain-scheduled design while directly synthesizing a globally stable, gain-scheduled multivariable controller.

6.3 Linear Parameter-Varying Simulation

The ROCETS nonlinear simulation is used to compare the baseline Pratt & Whitney controller with the LPV control designs. These simulations are performed with reference inputs to the power code, altitude and Mach number as a function of time. For some simulations, altitude and Mach number are held constant and only power code varied. Figure 6.8 shows the power code and altitude trajectory. Power code ranges from 3000 to 27000 and altitude varies from sea level to 40,000 ft. Figure 6.9 presents the candidate speed profile for the engine. Combined, Figures 6.8 and 6.9 represent a candidate stressful flight on the engine.

The LPV controllers were implemented in the ROCETS simulation as discussed in Chap-

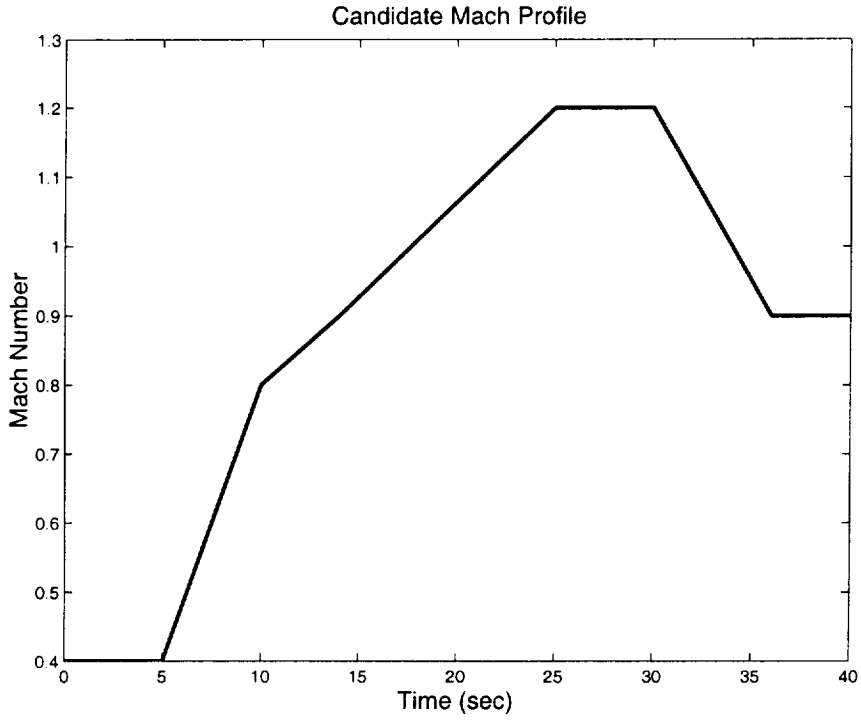

Figure 6.9: Speed as a function of time

ter 2. The LPV controllers are gain–scheduled as a function of a lagged measurement of power code via linear interpolation. The control trim inputs, based on the Jacobian linearization, added to the LPV control inputs are scheduled as a function of lagged power code and altitude. The nonlinear time responses of the LPV controllers are compared with the baseline controller. The comparison is useful from the standpoint that the performance and robustness of the baseline Pratt & Whitney multivariable controller was considered very good by Pratt & Whitney engineers. Note that the baseline Pratt & Whitney controller was scheduled as a scaled function of the air flow through the engine.

Figure 6.10 shows the response of the internal engine variables (OPR, EPR and N2) due to the baseline controller inputs (WFPRIB, AREANOZL and VANEHPC). Figure 6.11 shows normalized measurements, U1MVC, U2MVC, and U3MVC, provided to the baseline controller and the normalized outputs Y1MVC, Y2MVC, and Y3MVC. Note that in Figure 6.10 there is excellent tracking of OPR and EPR and as well as tracking of a lagged version of N2. These responses for the baseline for comparison with the LPV controllers.

Figure 6.12 shows the response of the internal engine variables (OPR, EPR and N2) with the non-rate bounded LPV controller and Figure 6.13 shows normalized measurements and outputs of the controller. Note the similarity between the baseline controller commands,

Figure 6.10, and the non-rate bounded LPV commands, Figure 6.12. Except for the small transient that occurs at 1 second due to switching between the baseline controller and the LPV controller, the engine response is nearly identical. The same excellent tracking of OPR and EPR and lagged N2 is achieved. It is interesting to see that the normalized controller Y1MVC and Y3MVC commands are very similar. Though for Y3MVC, the baseline controller commands a steady-state value of approximately 12 and the non-rate bounded LPV controller requests a steady-state value of approximately 2. It appears that this has little impact on the final N2 speed of the engine.

Figure 6.14 shows the response of the internal engine variables (OPR, EPR and N2) with the rate bounded LPV controller and Figure 6.15 shows normalized measurements and outputs of the controller. Note the similarity between the baseline controller commands, Figure 6.10, the non-rate bounded LPV commands, Figure 6.12, and the rate-bounded LPV commands, Figure 6.14. The same excellent tracking of OPR and EPR and lagged N2 is achieved. The non-rate bounded and the rate-bounded LPV controller have very similar control commands, even the steady-state value of Y3MVC, though the rate-bounded LPV controller has a a low bandwidth. This can be observed based on the reduced level of high frequency activity of the normalized control signals U1MVC and U2MVC (see Figures 6.13 and 6.15). The reduced bandwidth of the rate-bounded LPV controller is exactly what is expected with the introduction of bounds on the power code rate of variation.

The nonlinear ROCETS simulations with the LPV controllers implemented show that these controllers are robust to significant changes in the plant dynamics. Recall that the LPV controllers schedule only on lagged power code and were designed based on linearized engine models at zero Mach, sea level and standard day atmosphere. The candidate trajectory starts at sea level Mach 0.4 and reaches a peak altitude of 40,000 ft and speed of Mach 1.2. The performance of the LPV controllers is outstanding despite the large variation in speed and altitude. The effect of sensor noise on the control algorithms is also of interest. Therefore, sensor noise is added to the normalized measurements, U1MVC, U2MVC and U3MVC prior to the controller receiving these signals.

Figures 6.16 and 6.17 show the response of the engine and controller variables with a small amount (approximately 10%) of sensor noise on control input. The rate bounded LPV controller is implemented in the ROCETS nonlinear simulation. There is no degradation of the excellent tracking performance of the controller. Increasing the sensor noise to over 40%, shown in Figures 6.18 and 6.19 leads to some degradation of the tracking performance though the overall tracking performance is still adequate. Hence the rate-bounded LPV design is very insensitive to small amounts of sensor noise as well as changes in the plant dynamics. For comparison, Figures 6.20 and 6.21 show the response of the baseline controller with over 40% sensor noise. The baseline design is slightly more sensitive to the large amount of sensor

noise than the rate-bounded LPV controller.

Similarly, there is no change in the tracking performance of the rate-bounded LPV controller due to temperature fluctuation. Compare the response of the LPV controller on a standard day, Figures 6.14 and 6.15, with the response on a polar day, Figures 6.22 and 6.23, and a tropic day, Figures 6.24 and 6.25. Hence, the rate-bounded LPV controller is robust to significant variations.

The tracking performance of the LPV controller is perhaps better illustrated by holding the speed constant at zero Mach, standard atmosphere and a constant non-zero altitude. Power code is varied as a function of time. Figures 6.26 and 6.27 show the response of the non-rate bounded LPV controller at sea level. This compares well with the rate-bounded LPV design at sea level, Figures 6.28 and 6.29. Figures 6.30, 6.31, 6.32 and 6.33 show the engine and rate bounded LPV controller time responses at 15,000 and 30,000 ft altitude respectively. It is obvious that the engine dynamics change dramatically as a function of altitude.

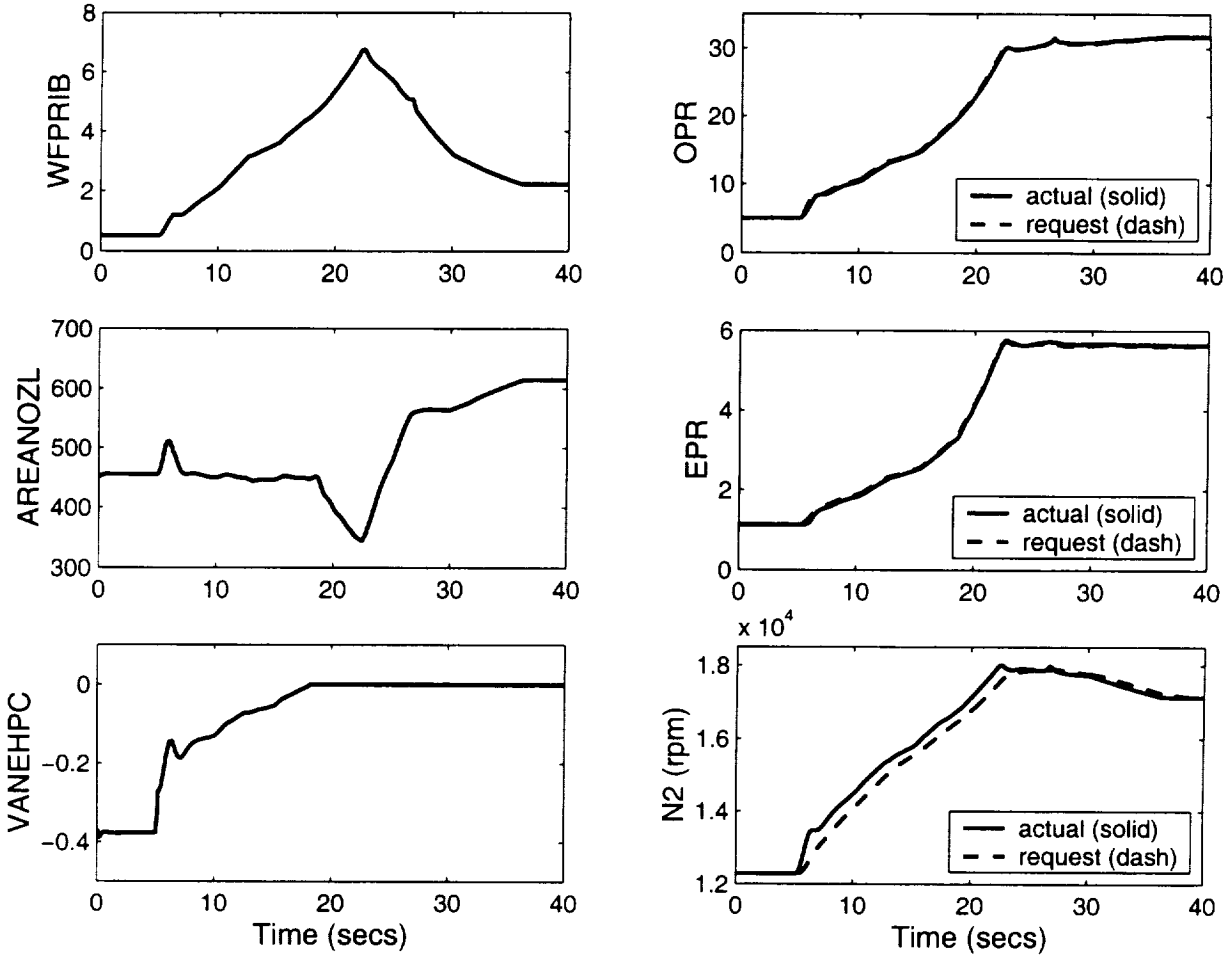

Figure 6.10: Engine response for candidate flight: Baseline Controller

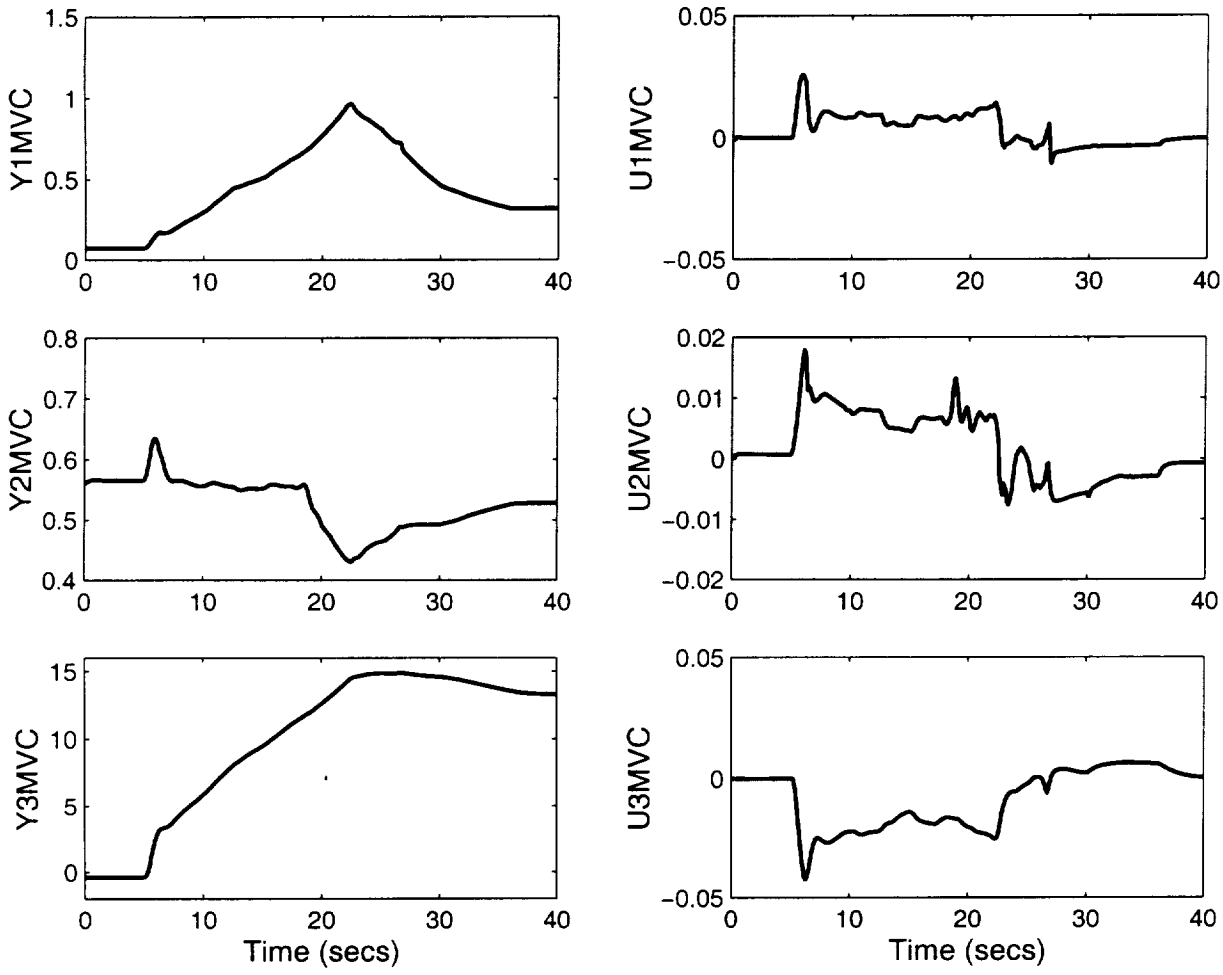

Figure 6.11: Controller input and output responses for candidate flight: Baseline Controller

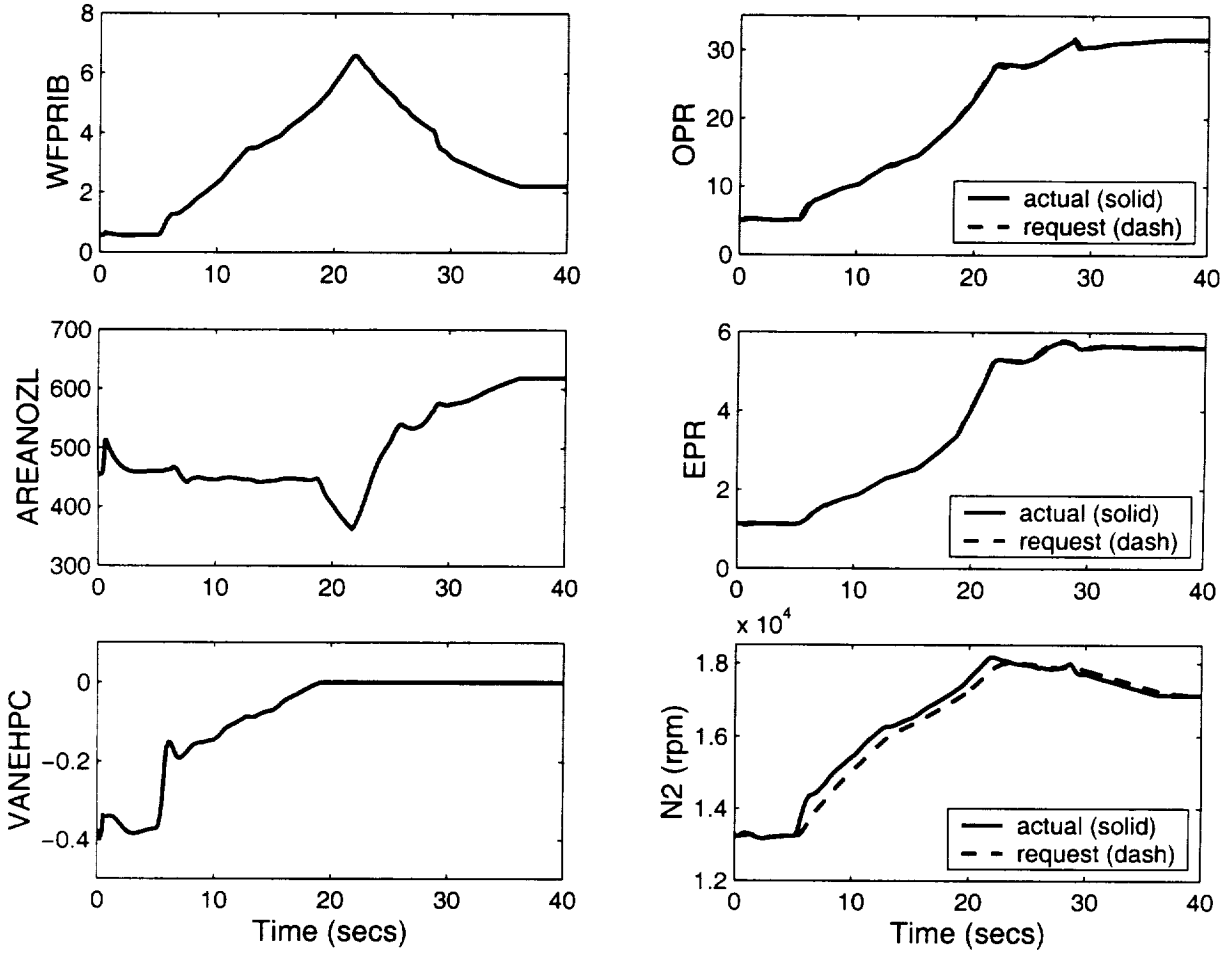

Figure 6.12: Engine response for candidate flight: Non-Rate Bounded LPV Controller

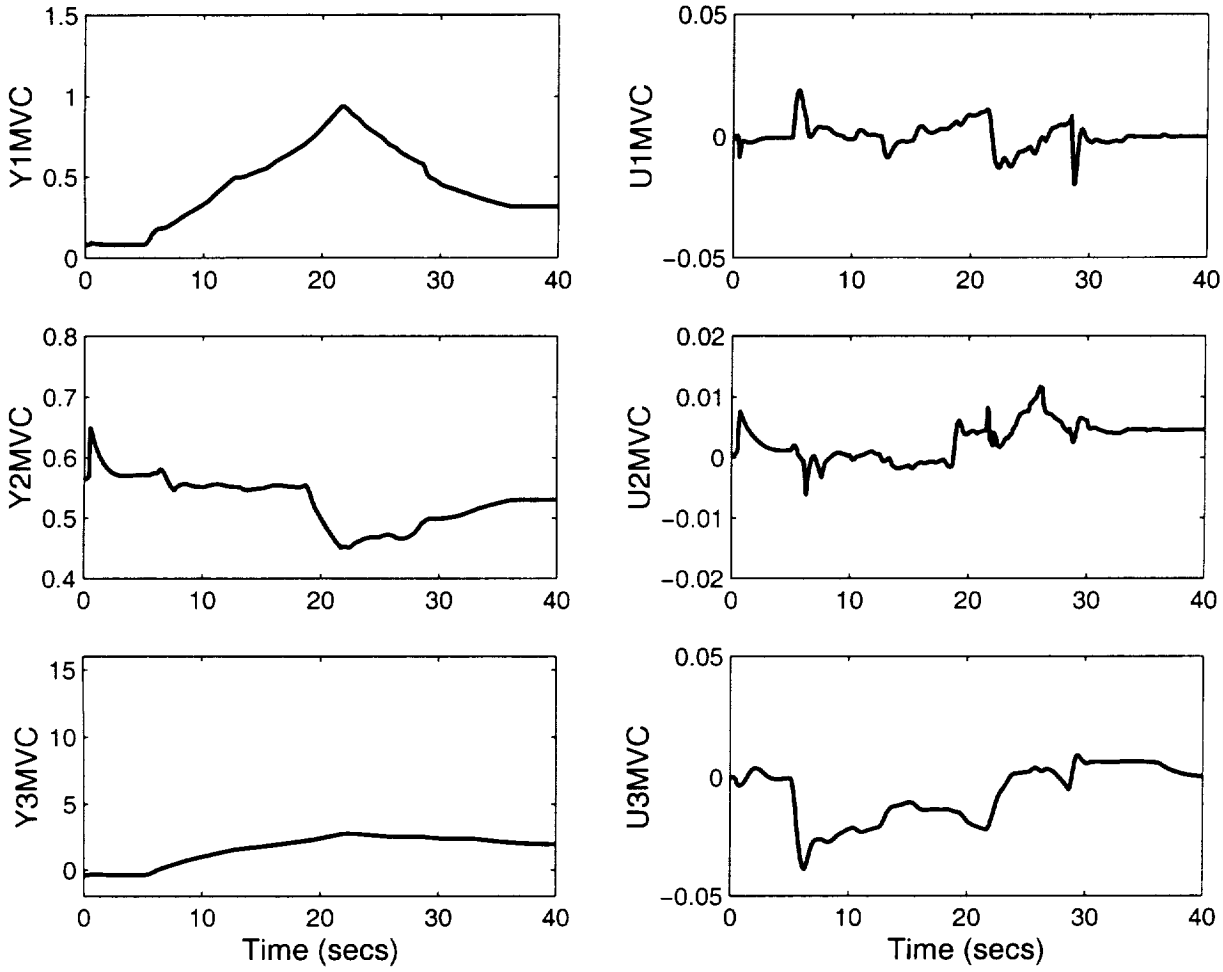

Figure 6.13: Controller input and output responses for candidate flight: Non-Rate Bounded LPV Controller

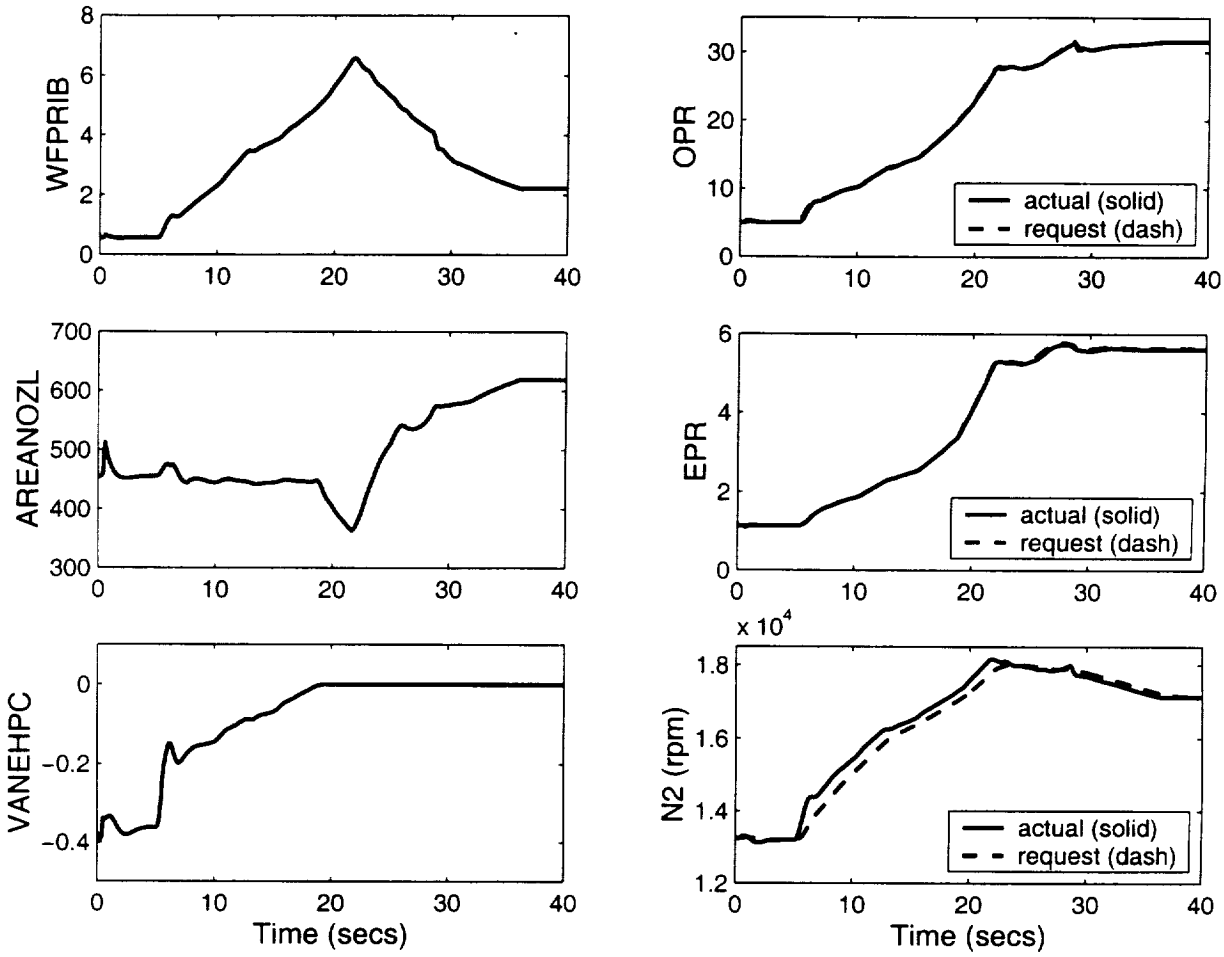

Figure 6.14: Engine response for candidate flight: Rate Bounded LPV Controller

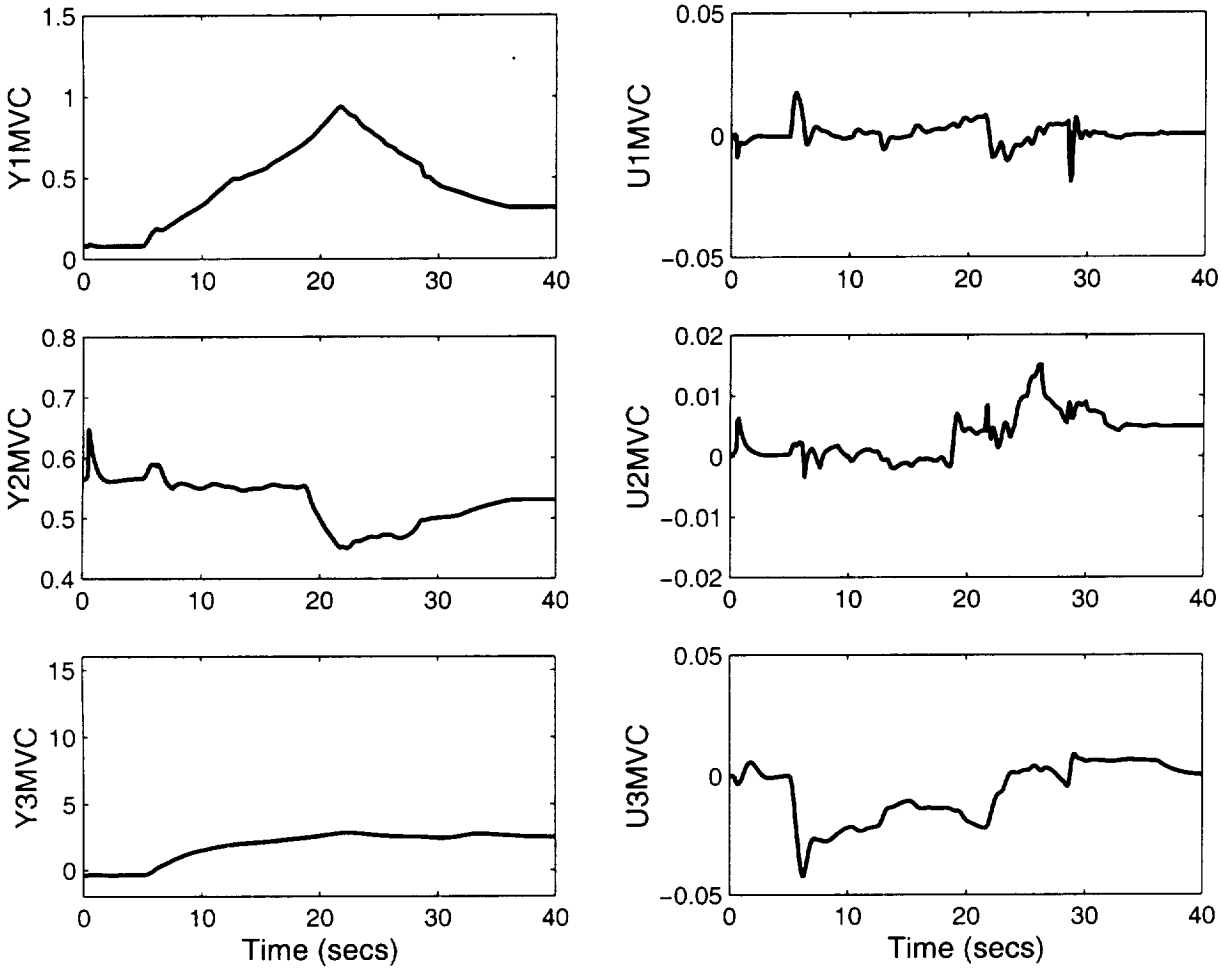

Figure 6.15: Controller input and output responses for candidate flight: Rate Bounded LPV Controller

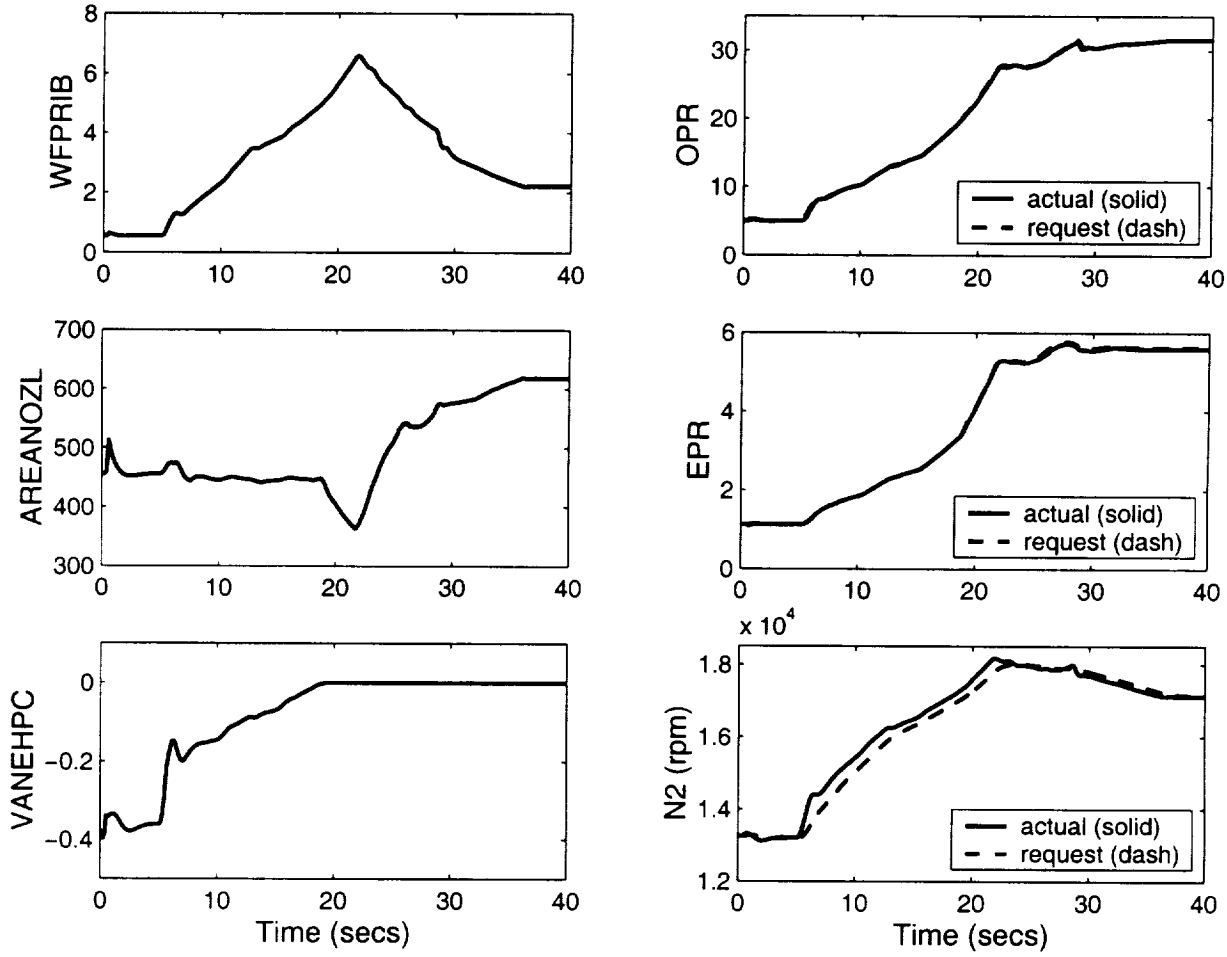

Figure 6.16: Engine response for candidate flight with small sensor noise: Rate Bounded LPV Controller

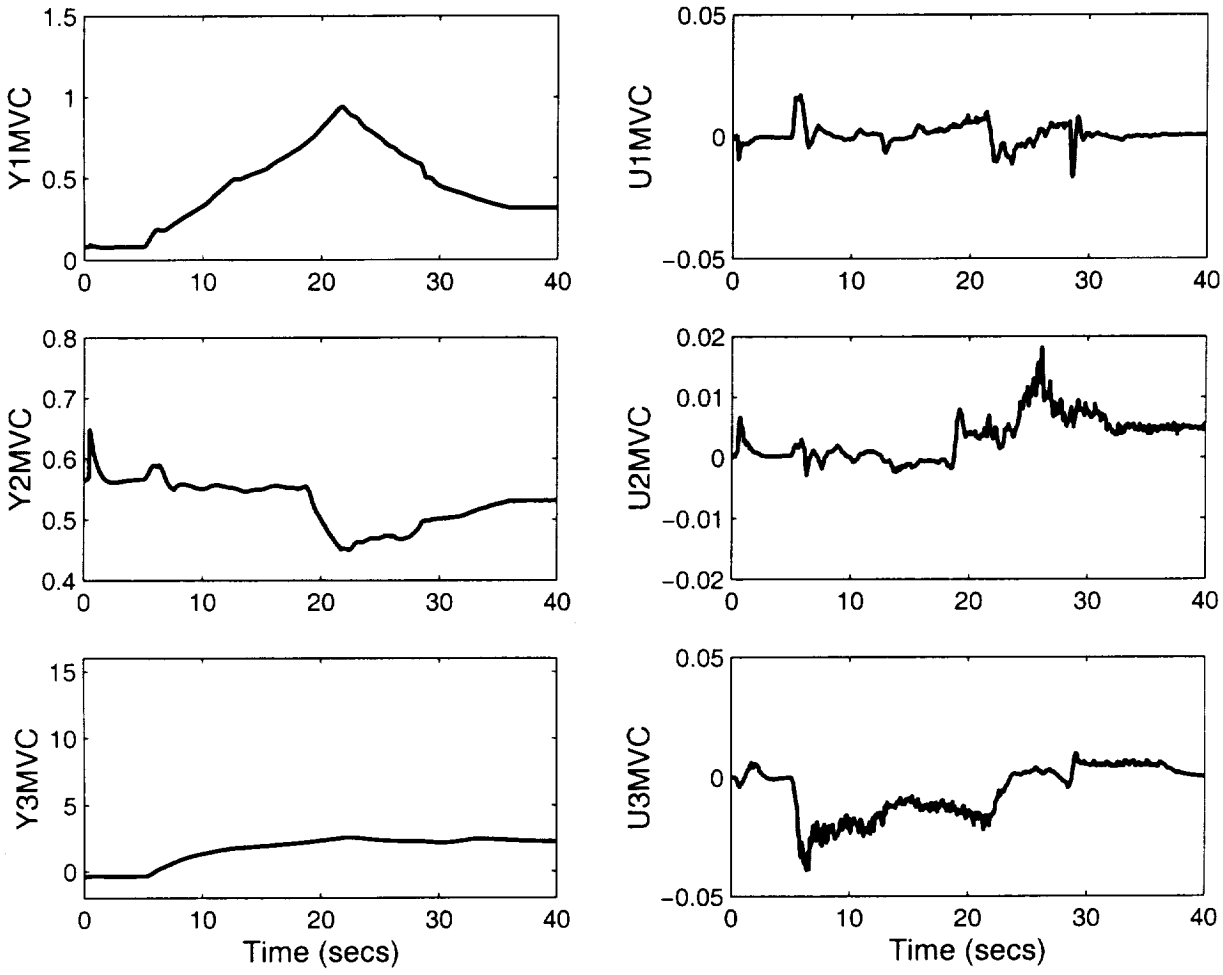

Figure 6.17: Controller input and output responses for candidate flight with small sensor noise: Rate Bounded LPV Controller

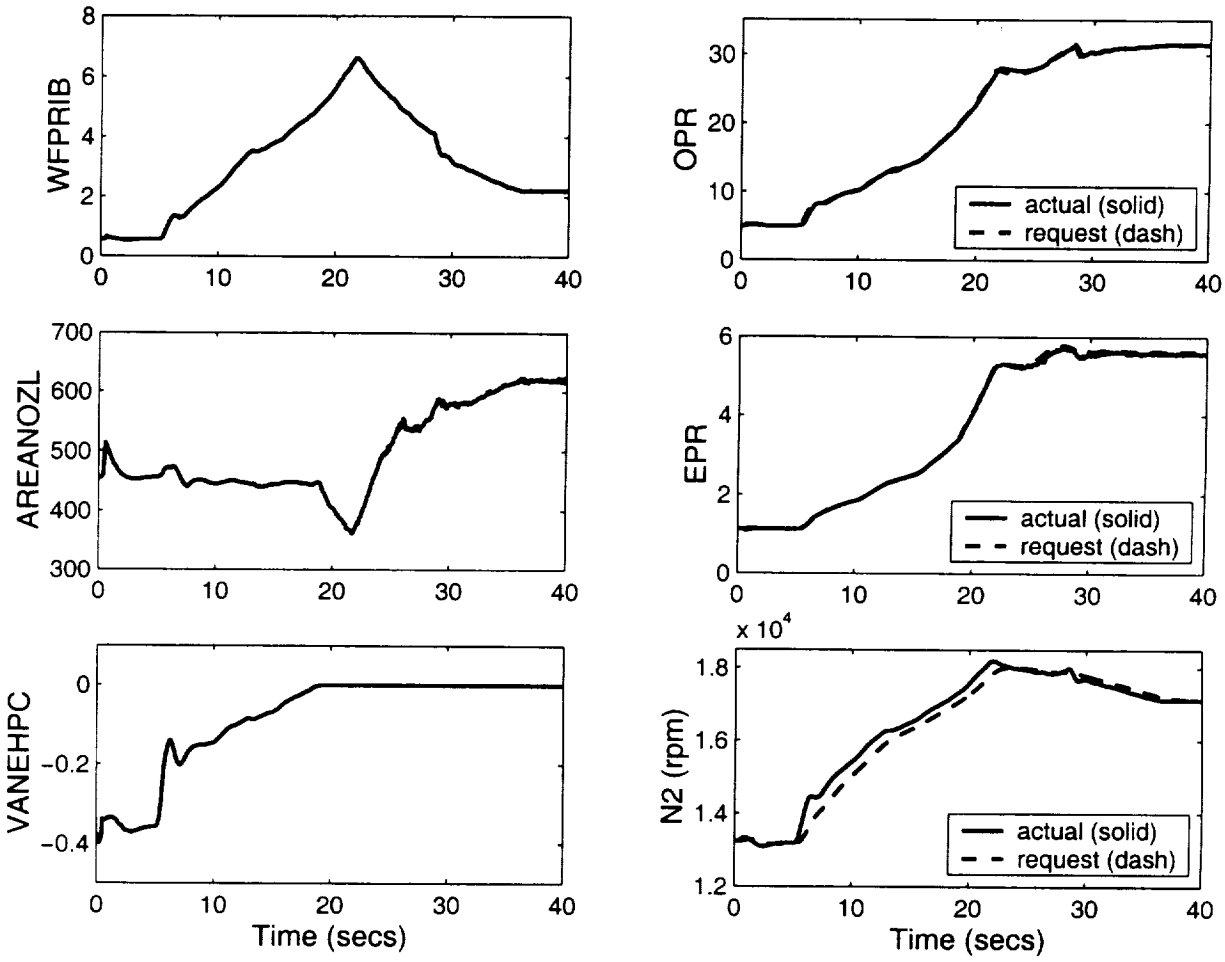

Figure 6.18: Engine response for candidate flight with large sensor noise: Rate Bounded LPV Controller

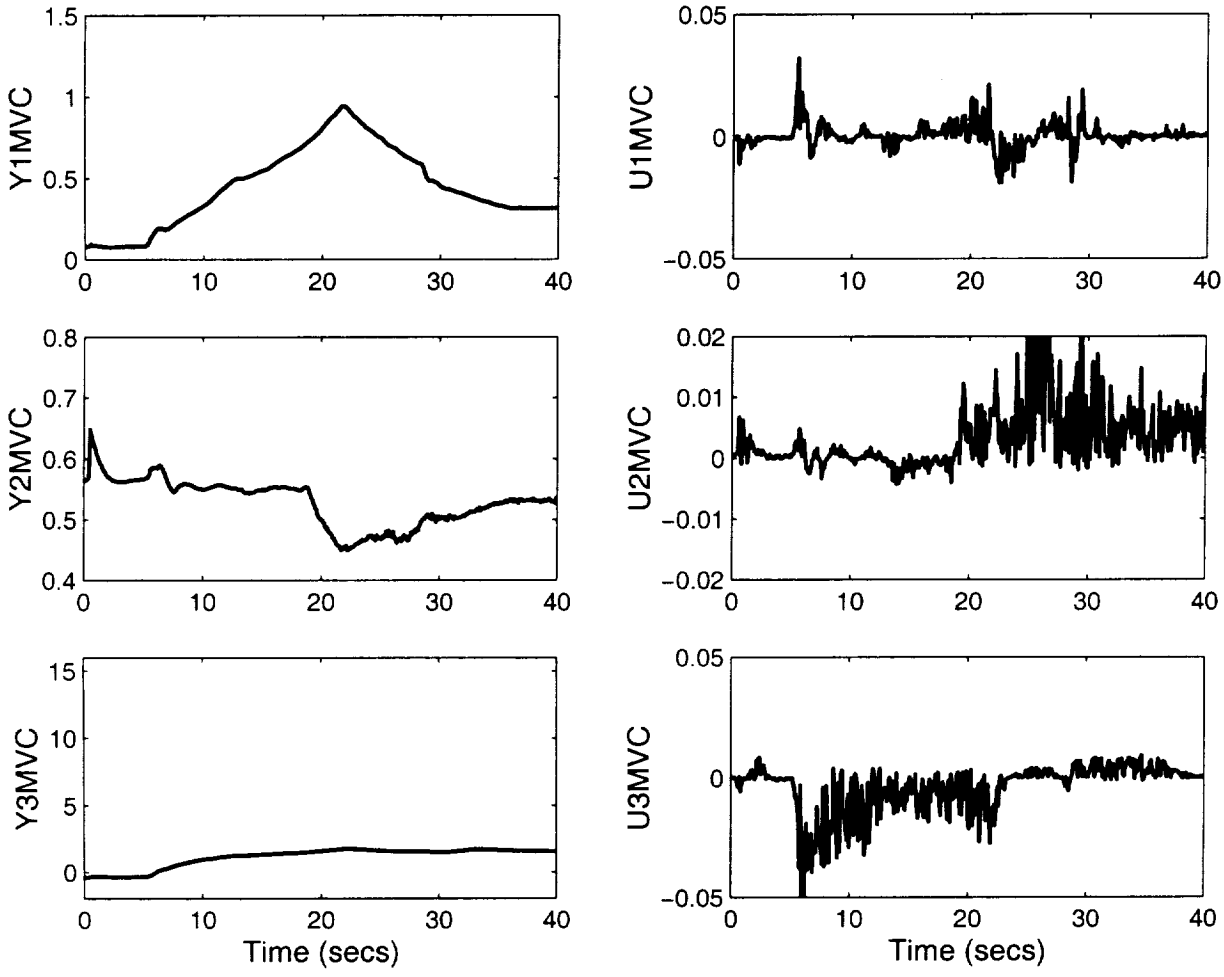

Figure 6.19: Controller input and output responses for candidate flight with large sensor noise: Rate Bounded LPV Controller

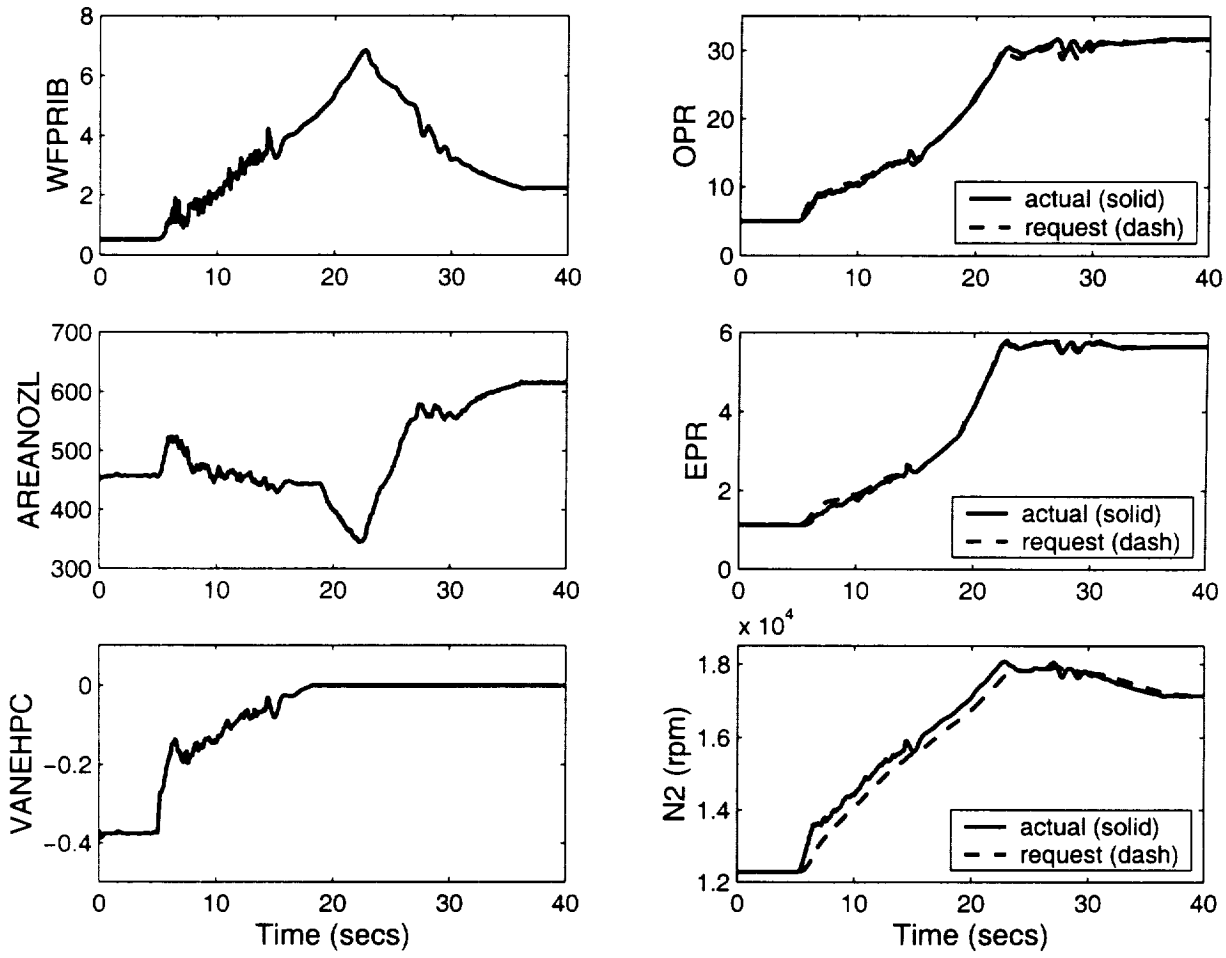

Figure 6.20: Engine response for candidate flight with large sensor noise: Original Controller

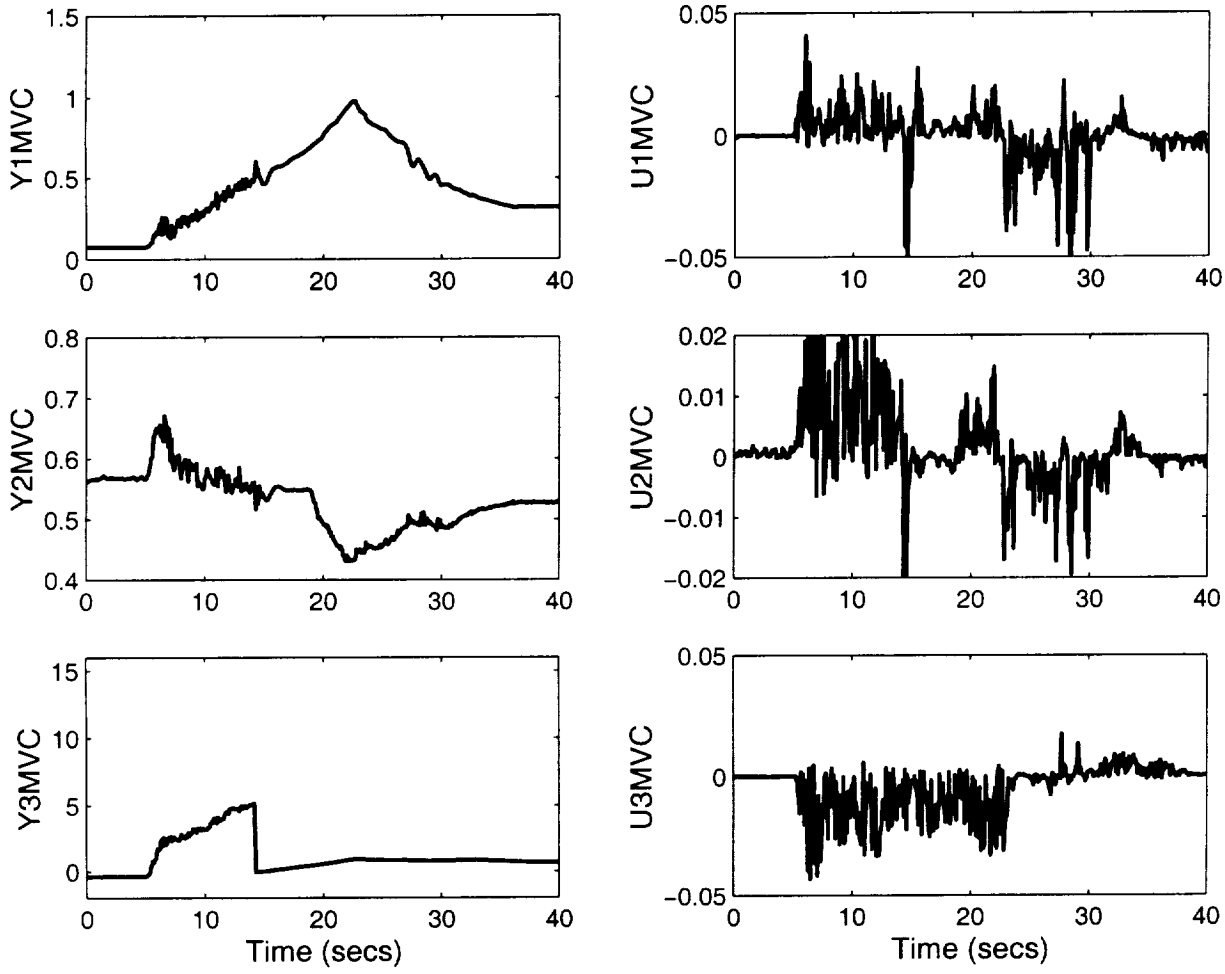

Figure 6.21: Controller input and output responses for candidate flight with large sensor noise: Original Controller

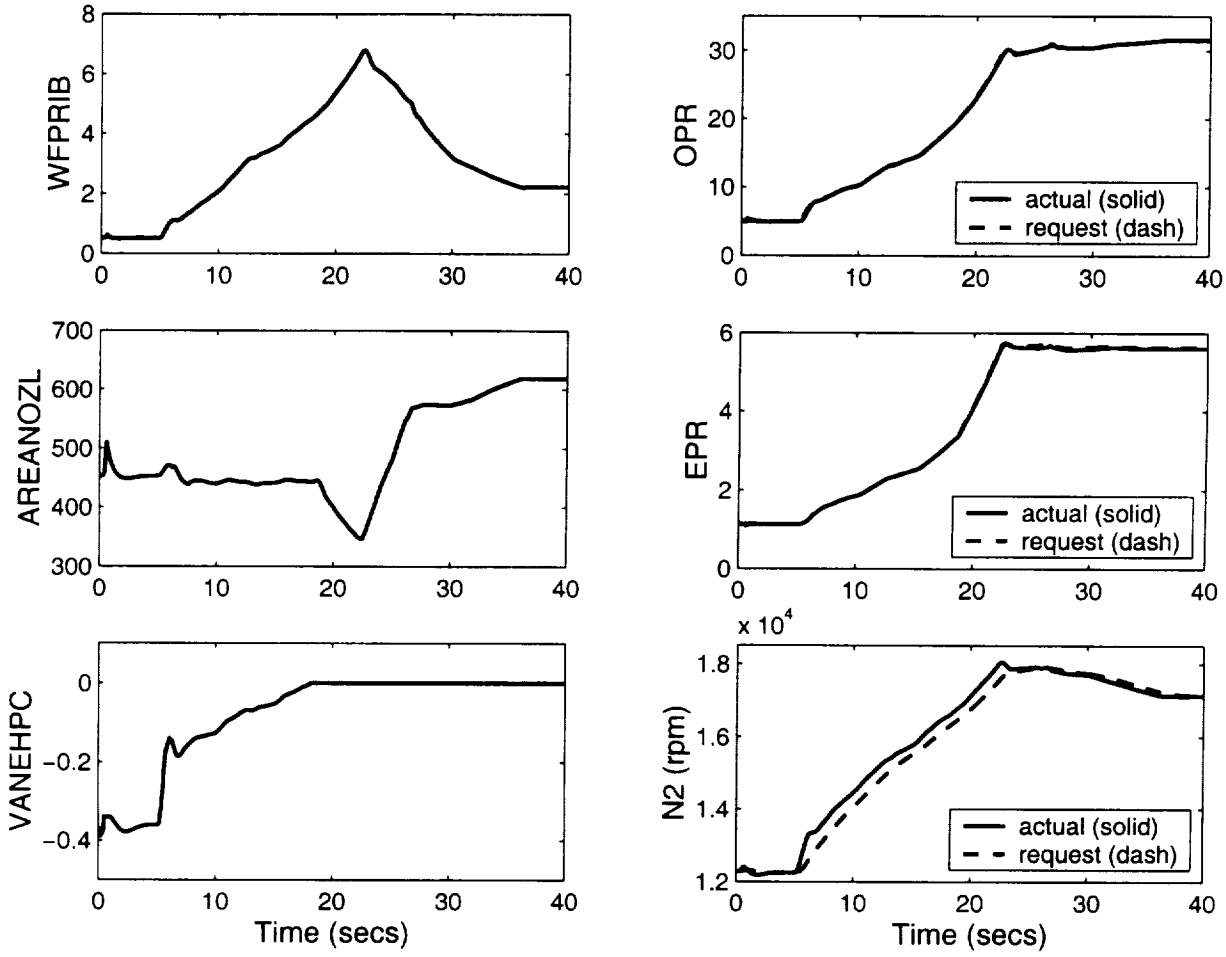

Figure 6.22: Engine response for candidate flight polar day: Rate Bounded LPV Controller

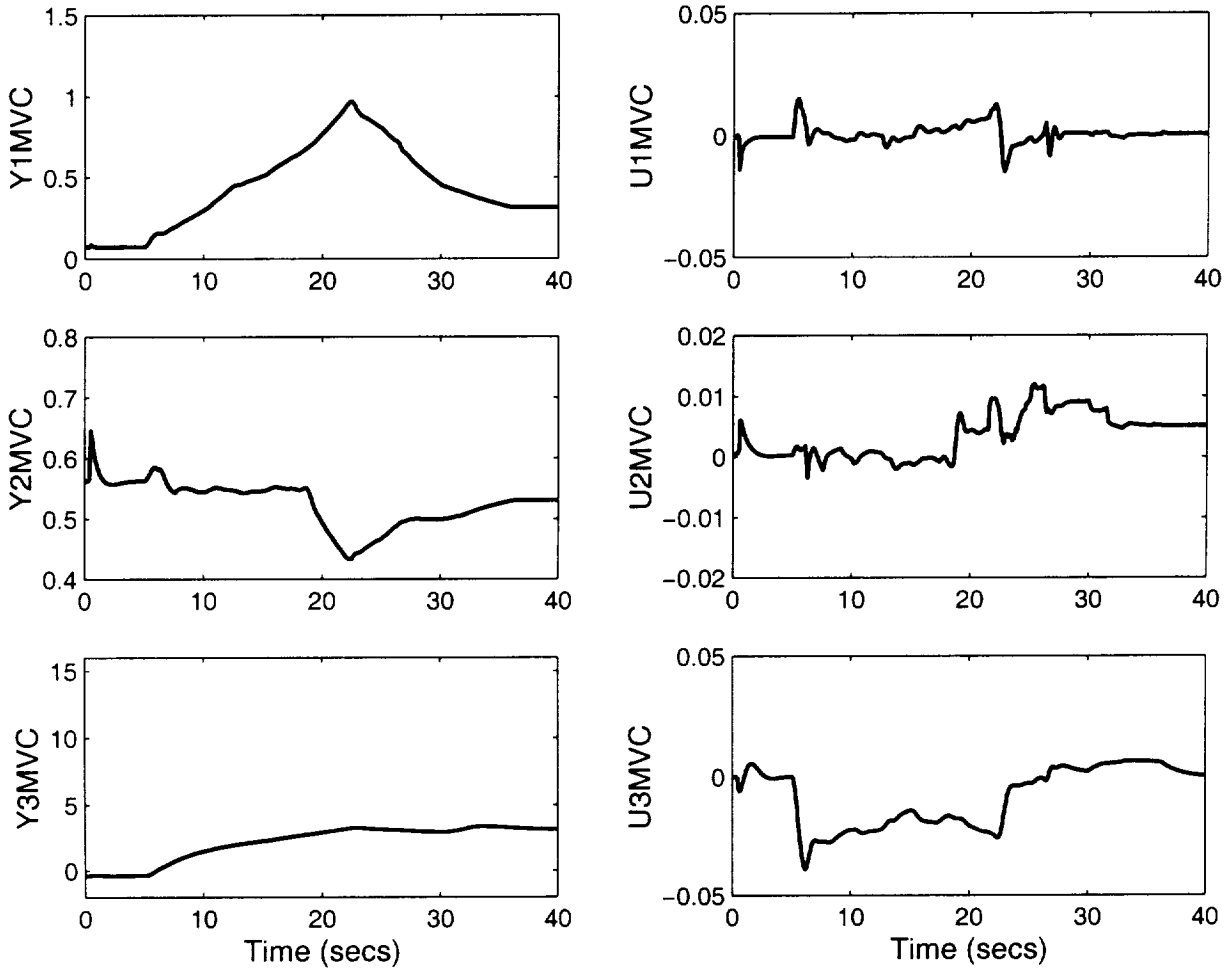

Figure 6.23: Controller input and output responses for candidate flight polar day: Rate Bounded LPV Controller

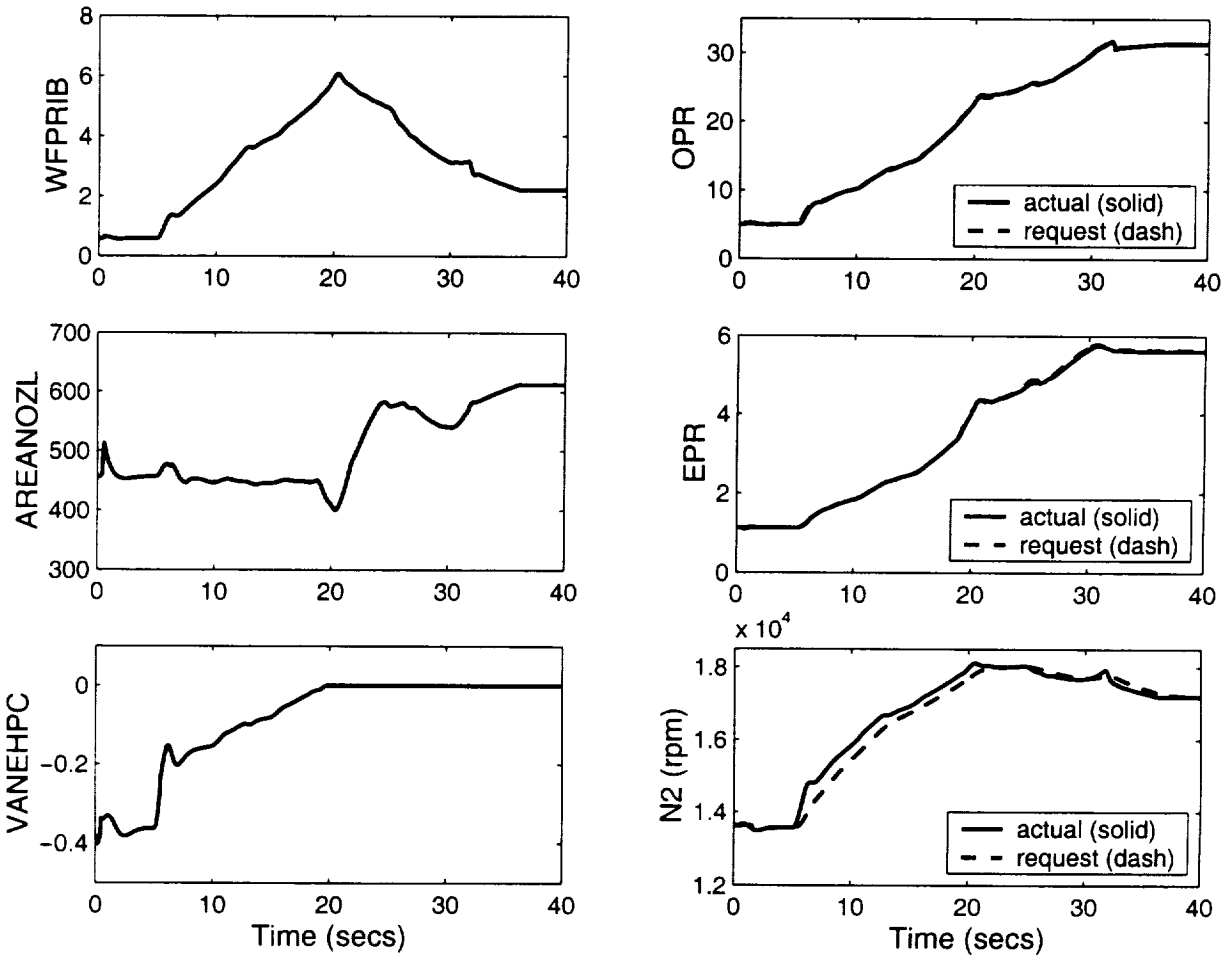

Figure 6.24: Engine response for candidate flight tropical day: Rate Bounded LPV Controller

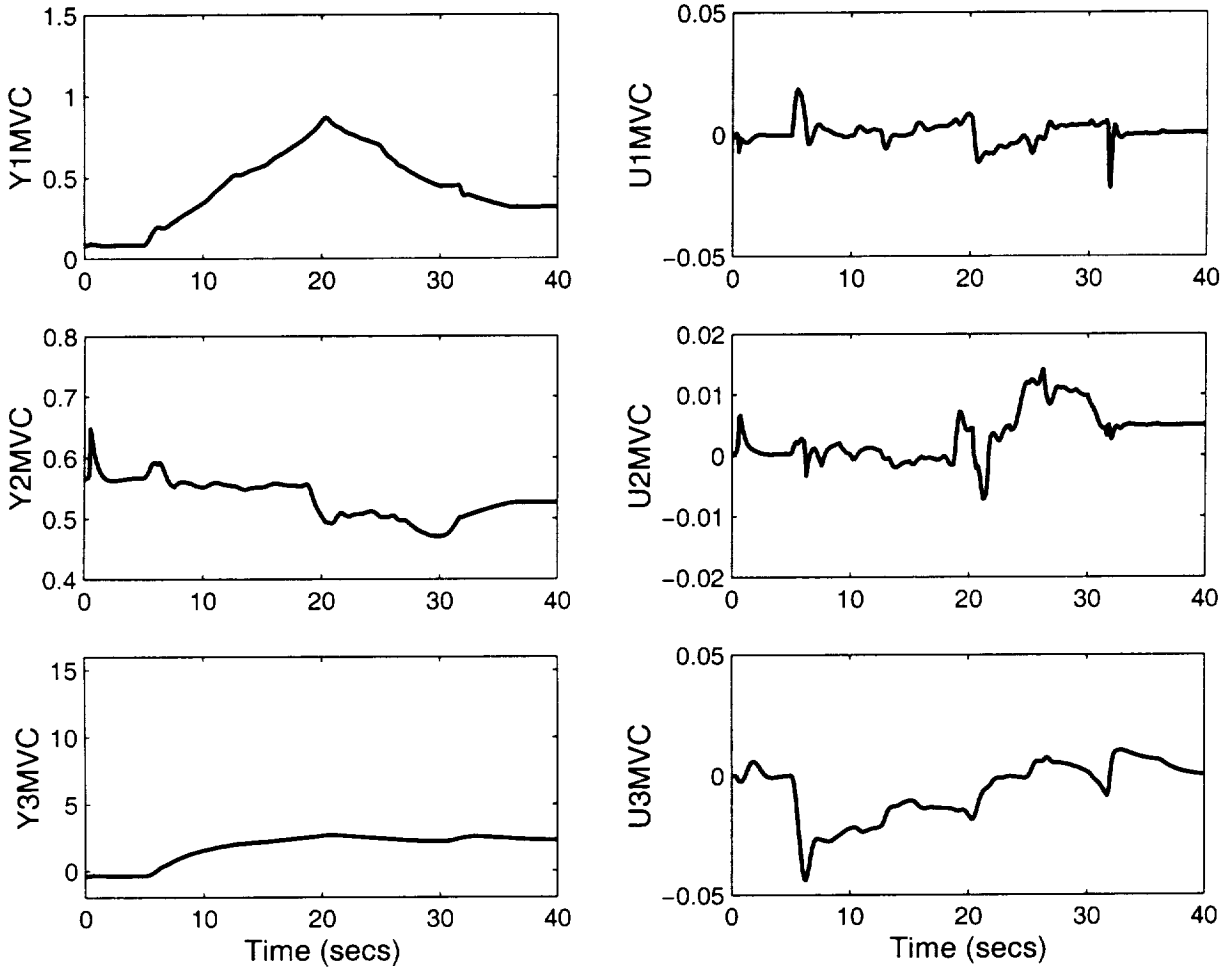

Figure 6.25: Controller input and output responses for candidate flight tropical day: Rate Bounded LPV Controller

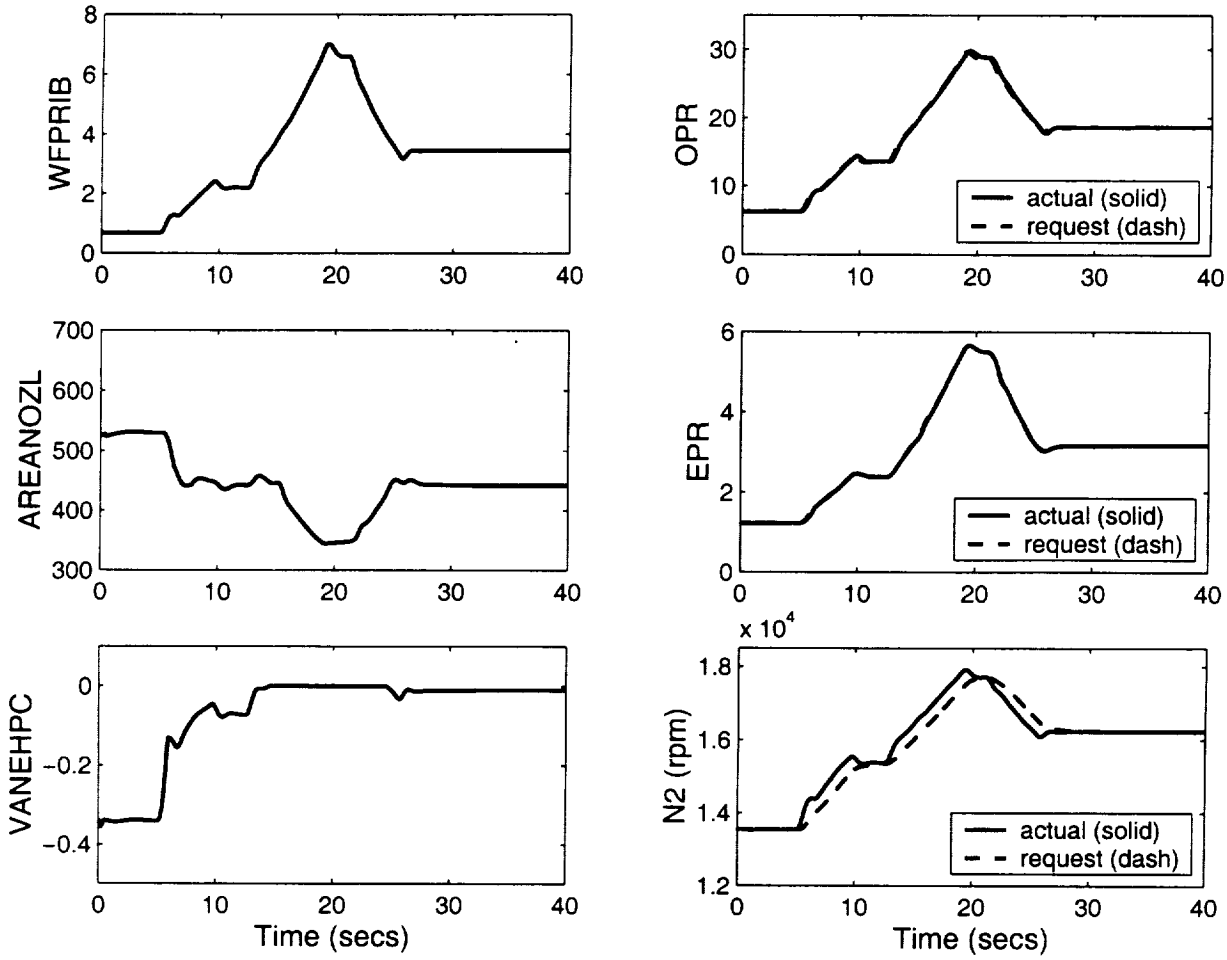

Figure 6.26: Engine response for sea level flight: Non-Rate Bounded LPV Controller

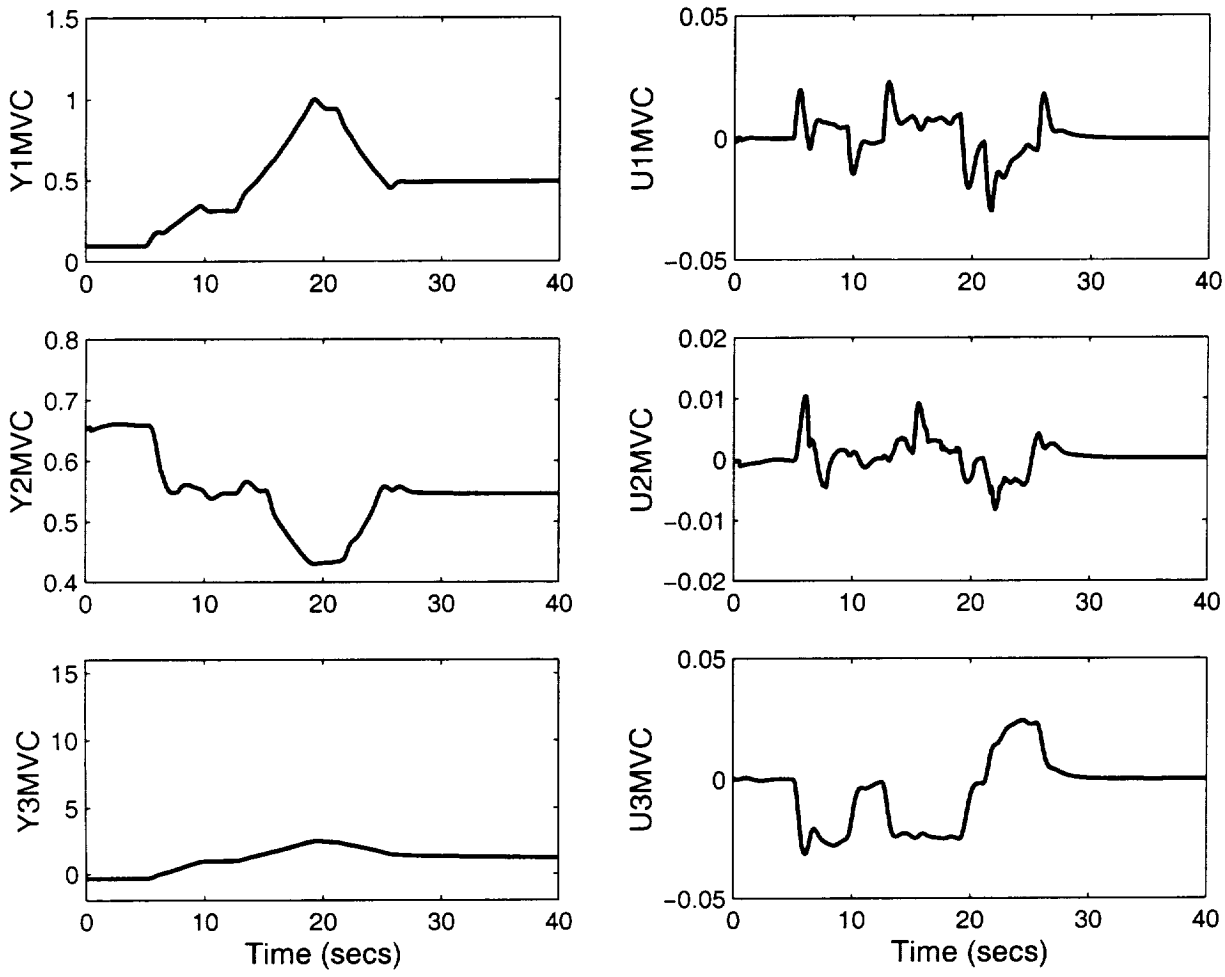

Figure 6.27: Controller input and output responses for sea level flight: Non-Rate Bounded LPV Controller

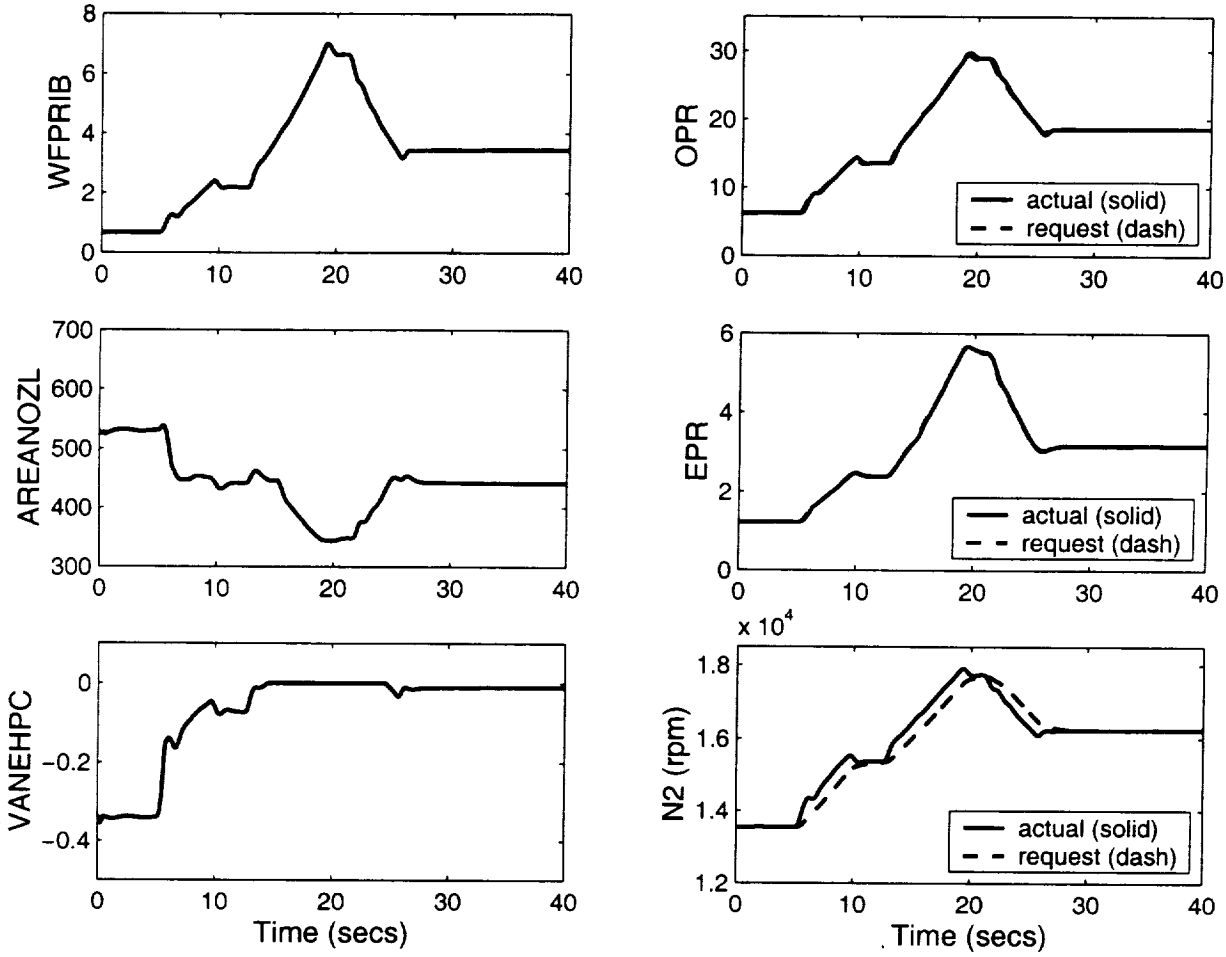

Figure 6.28: Engine response for sea level flight: Rate Bounded LPV Controller

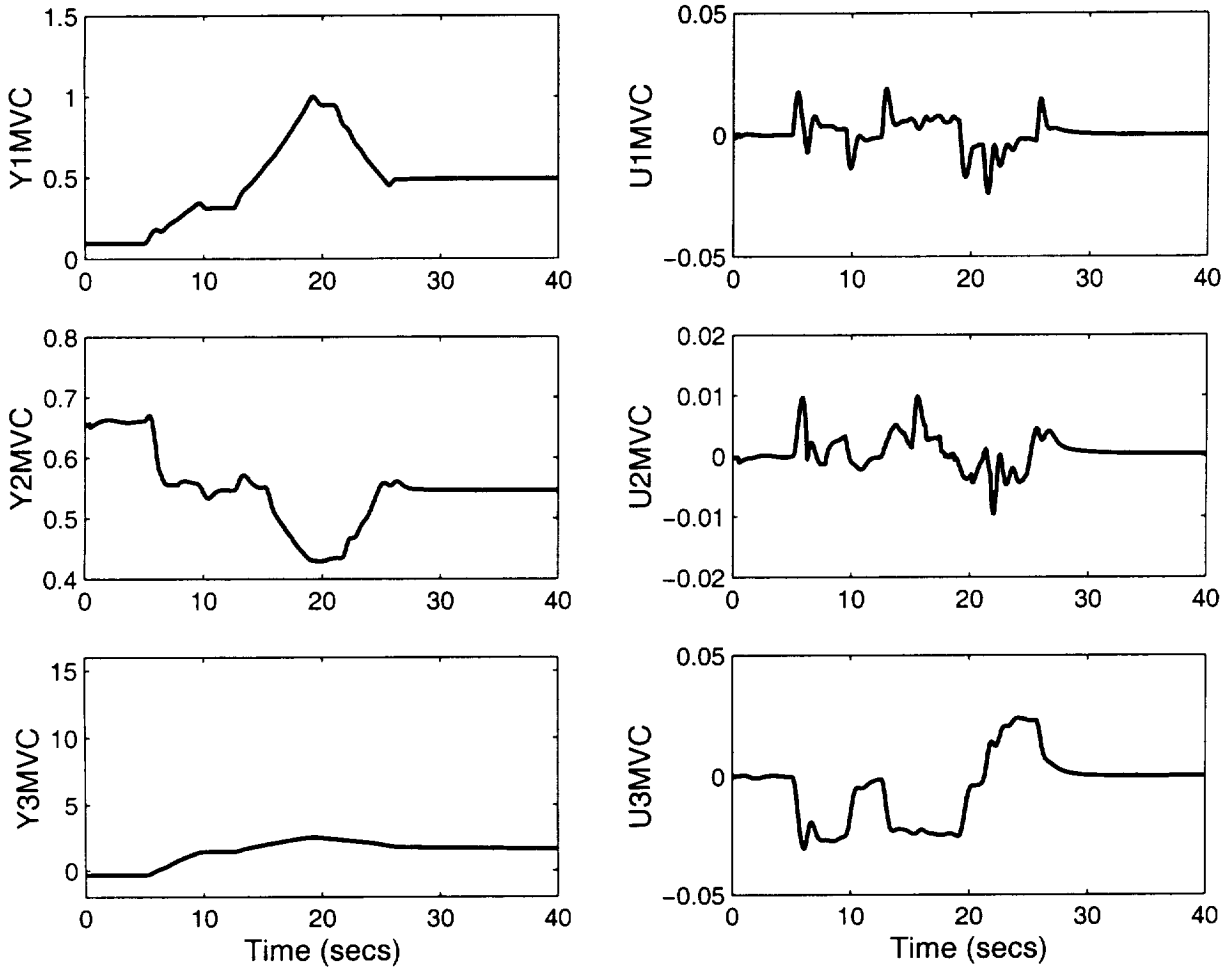

Figure 6.29: Controller input and output responses for sea level flight: Rate Bounded LPV Controller

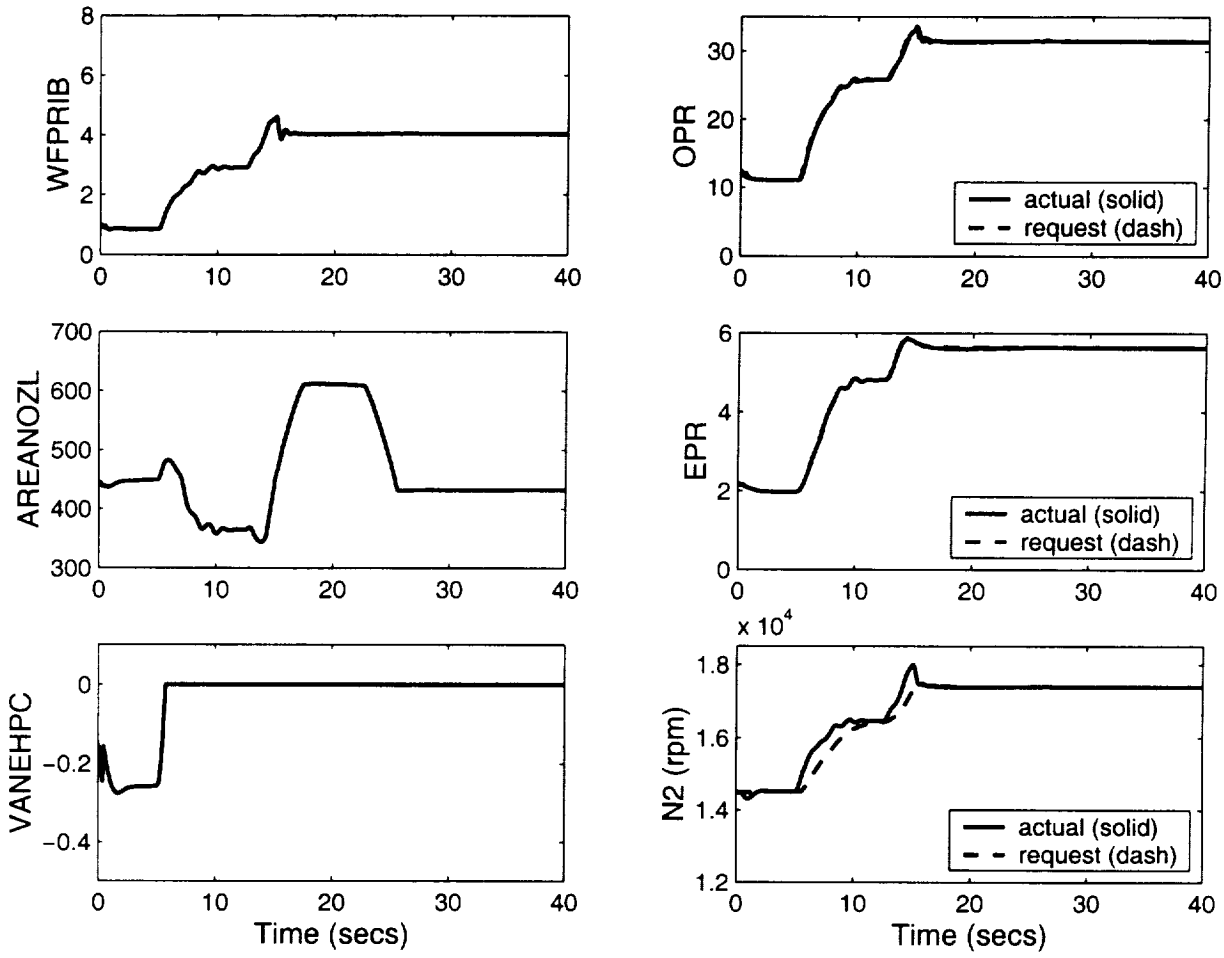

Figure 6.30: Engine response for 15K ft altitude flight: Rate Bounded LPV Controller

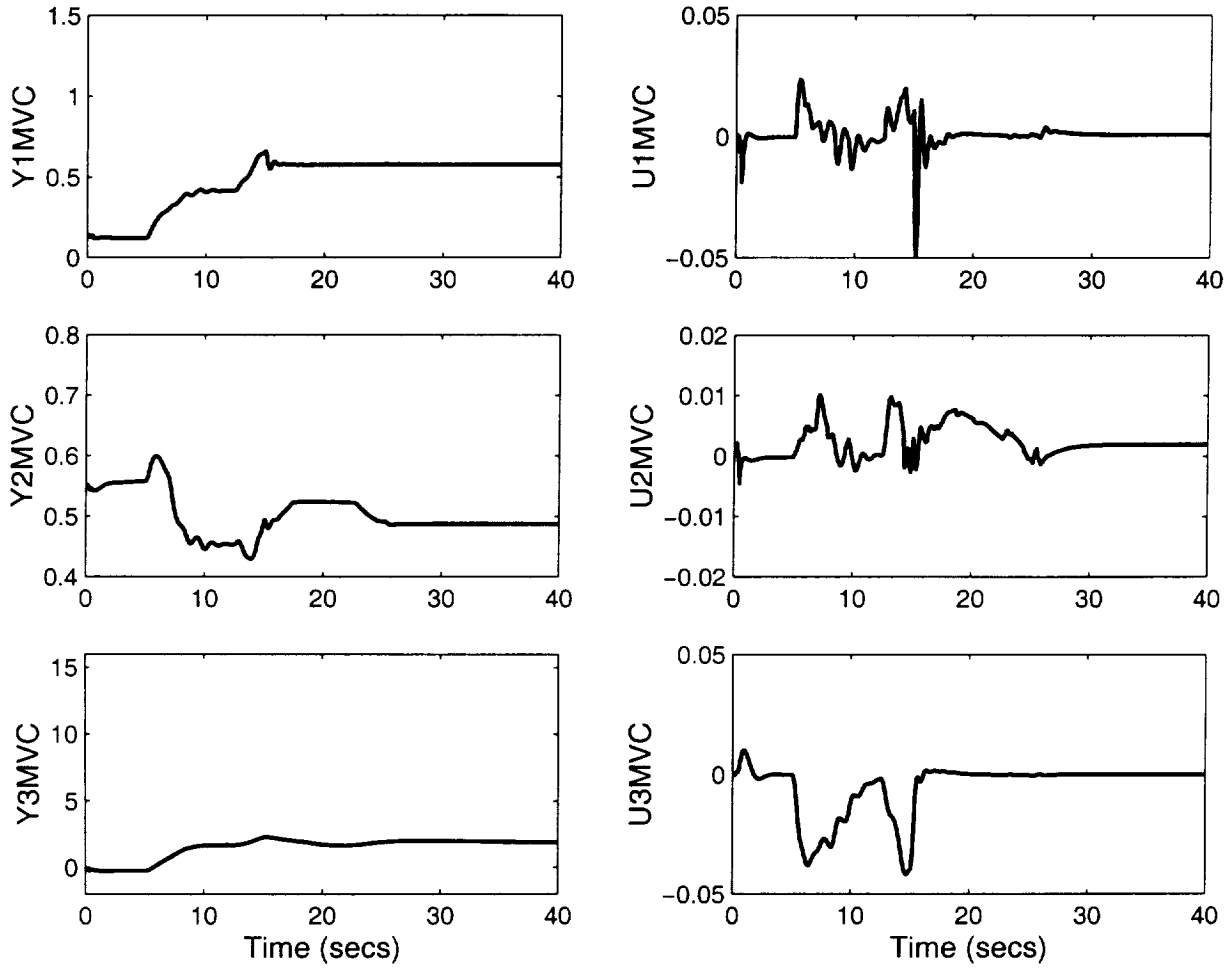

Figure 6.31: Controller input and output responses for 15K ft altitude flight: Rate Bounded LPV Controller

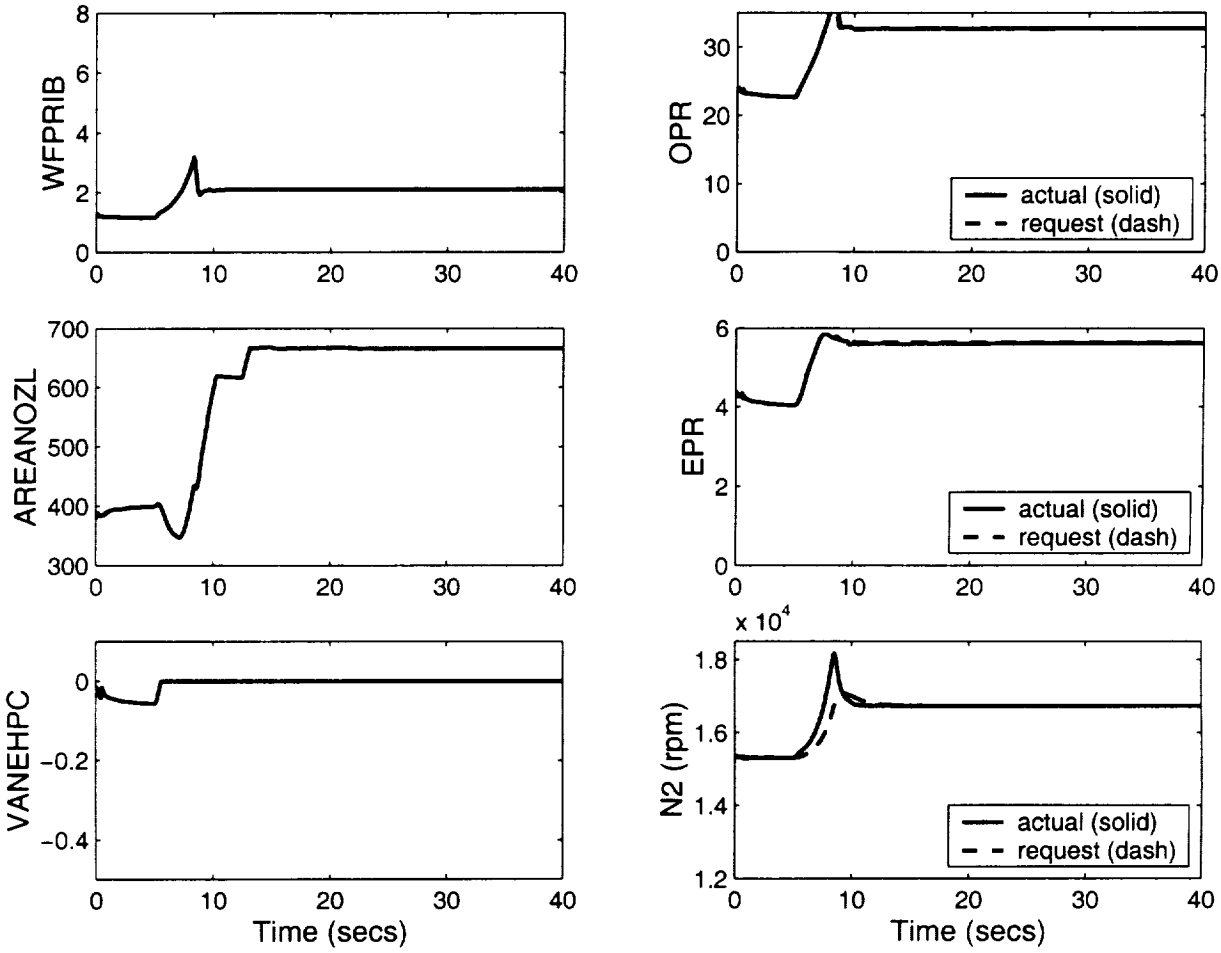

Figure 6.32: Engine response for 30K ft altitude flight: Rate Bounded LPV Controller

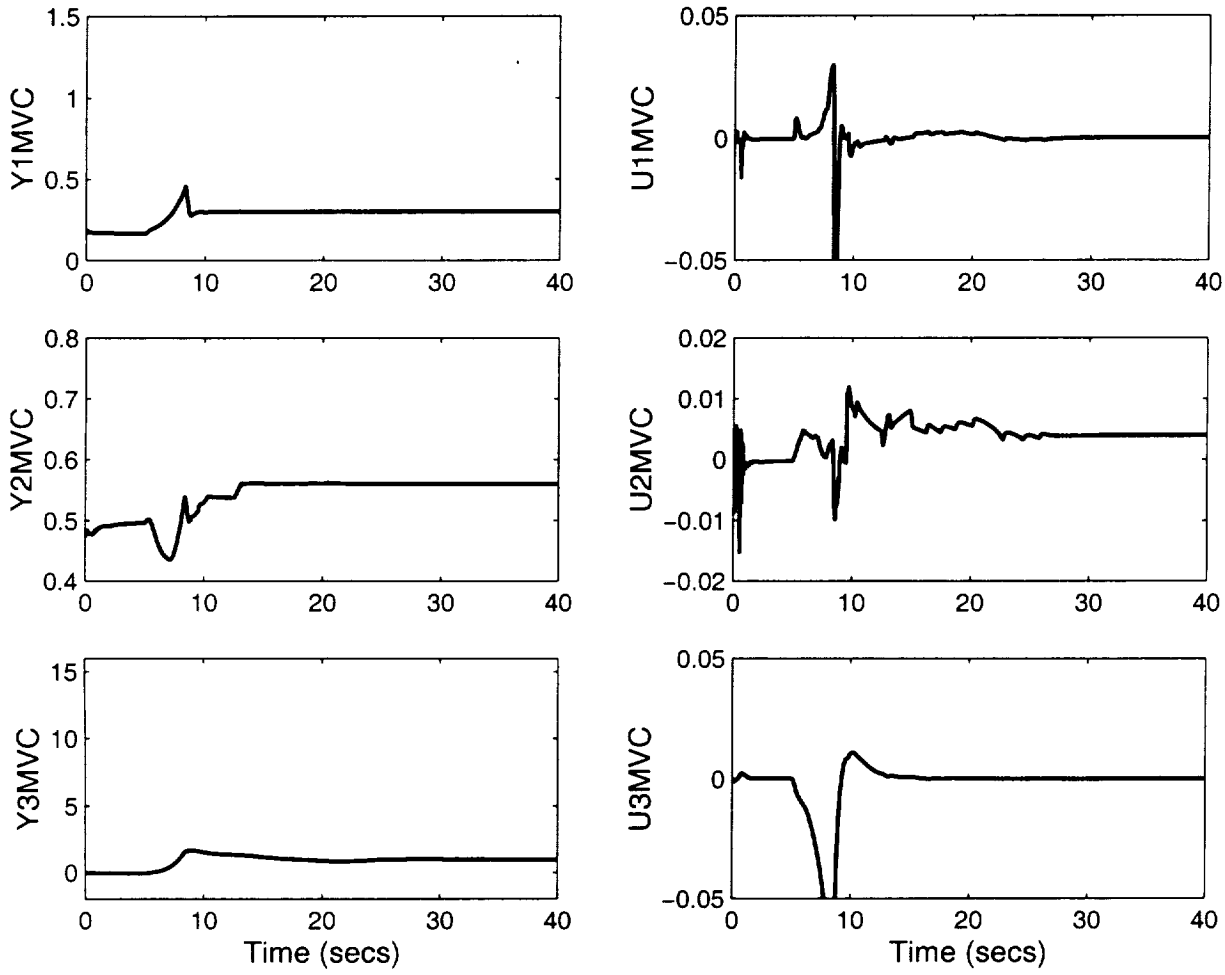

Figure 6.33: Controller input and output responses for 30K ft altitude flight: Rate Bounded LPV Controller

Chapter 7

Summary

Recent advances in robust control synthesis for LPV systems were applied to a high fidelity, nonlinear simulation of a turbofan engine under this research program. The control objective was to decouple the multivariable system into three independent channels with minimal cross–coupling, subject to actuator magnitude and rate limitations. $\mathcal{H}_\infty$ point designs provided an initial starting point, giving an indication of the performance which might be expected of an LPV controller. After obtaining reasonable point designs, an LPV controller was synthesized and its performance evaluated.

The nonlinear ROCETS simulation of the Pratt & Whitney engine with the LPV controllers implemented achieved excellent tracking of the command signals with small magnitude actuator commands. The rate-bounded LPV controller were robust to significant variations in the engine model dynamics, atmospheric conditions and sensor noise. Excellent tracking performance was achieved it all scenarios except in the presence of large sensor noise. Although the synthesis of LPV controllers are computationally more intensive than individual linear point designs, the guaranteed and actual performance and robustness of the gain-scheduled LPV designs indicate the benefit of the rate-bounded LPV control designs far out weighs their computation time expense.

7.1 Acknowledgements

This work was funded by NASA Lewis Research Center, Contract No. NASA/NAG3-1975 with Jonathan Litt as the contract monitor. The authors would like to thank Steve Watts and Randy Rosson from Pratt & Whitney and Sanjay Garg from NASA Lewis for help with the ROCETS code and for insight into the design of multivariable controllers for turbofan engines.

Bibliography

[1] Pratt & Whitney, CCD 1453-00.0. User's manual for SCIP transient engine simulation. FR-21407B, Aeropropulsion and Power Directorate, Wright Laboratory. Wright-Patterson AFB, Ohio, Dec. 20, 1996.

[2] D. K. Frederick, S. Garg, and S. Adibhatla. Turbofan engine control design using robust multivariable control technologies. In *AIAA, ASME, SAE, and ASEE Joint Propulsion Conference*, Lake Buena Vista, FL, 1996. AIAA paper # 96–2587.

[3] A. Packard. Gain scheduling via linear fractional transformations. *Systems and Control Lett.*, 22(2):79–92, 1994.

[4] A. Packard and M. Kantner. Gain scheduling the LPV way. In *Proc. 35th IEEE CDC*, pages 3938–3941, Kobe, Japan, 1996.

[5] G. Becker. *Quadratic stability and performance of linear parameter dependent systems.* PhD thesis, University of California, Berkeley, 1993.

[6] F. Wu, A. Packard, and G.J. Balas. LPV control design for pitch-axis missile autopilots. In *Proc. 34th IEEE CDC*, pages 188–193, New Orleans, LA, 1995.

[7] F. Wu, X.H. Yang, A. Packard, G. Becker. Induced L_2-norm control for LPV system with bounded parameter variation rates. in *Proc. Amercian Control Conference*, pages 2379-2383, Vol. 3, Seattle, WA, 1995.

[8] G. J. Balas, I. J. Fialho, A. K. Packard, J. Renfrow, and C. Mullaney. On the design of LPV controllers for the F-14 lateral–directional axis during powered approach. In *Proc. 1997 ACC*, Albuquerque, NM, 1997.

[9] E. Kamen and P. Khargonekar. On the control of linear systems whose coefficients are functions of parameters. *IEEE Trans. Auto. Control*, AC-29(1):25–33, 1984.

[10] S. M. Shahruz and S. Behtash. Design of controllers for linear parameter-varying systems by the gain scheduling technique. In *Proc. 29th IEEE CDC*, pages 2490–2491, Honolulu, HI, 1990.

[11] J. Shamma and M. Athans. Gain scheduling: potential hazards and possible remedies. In *Proc. 1991 ACC*, pages 516–521, Boston, MA, 1991.

[12] P. Apkarian and P. Gahinet. A convex characterization of gain-scheduled H_∞ controllers. *IEEE Trans. Auto. Control*, AC-40(5):853–864, 1995.

[13] G. J. Balas, J. C. Doyle, K. Glover, A. Packard, and R. Smith. *μ-Analysis and Synthesis Toolbox (μ-Tools).* The MathWorks, Inc., Natick, MA, 1991.

[14] A. K. Packard and J. C. Doyle. The complex structured singular value. *Automatica*, 29(1):71–109, 1993.

[15] G. J. Balas, Jack Ryan, Jong-Yeob Shin, William Garrard. A New Technique for Design of Controllers for Turbofan Engines. In *AIAA, ASME, SAE, and ASEE Joint Propulsion Conference*, Cleveland, OH, 1998. AIAA paper # 98–3751.

[16] K. Zhou, J.C. Doyle, K. Glover. Robust and Optimal Control. *Prentice Hall*, 1995.

[17] W. Tan, A.K. Packard and G.J. Balas. Quasi-LPV modeling and LPV control of a generic missile *2000 American Control Conference*, Chicago, IL, June, 2000, pp. 3692-3696.

[18] J.C. Doyle, K. Glover, P.P. Khargonekar, and B.A. Francis. State-space solutions to standard H_2 and H_∞ control problems. *IEEE Trans. Auto. Control*, vol. AC-34, no. 8, August 1989, pp 831-847.

Appendix A

User's Guide

A.1 Introduction

This Appendix is intended to be used as a user's manual for the UNIX script files, Matlab M files, and the modified SCIP transient engine simulations used in this work. The reader interested in details of the SCIP transient engine simulation is directed to [1].

A.2 Nonlinear Simulation

To simplify the operation of the nonlinear simulation, a series of UNIX script files were written. They run the correct files and rename the input and output files more descriptively then the simulation does alone. The file `runorig` runs the original program provided to the University or Minnesota. The file `runptdes` reads in a point-design controller to replace the Pratt & Whitney controller in the simulation. The file `runlpv` reads in a LPV controller to replace the Pratt & Whitney controller. It is assumed in the following discussions that these script files are being used. These script files are discussed further in Section A.4.

The original and modified versions of the SCIP transient engine simulation read an input file containing all run parameters, and generate four output files. The input file must end with the extension ".dat". For example `filename.dat` is an acceptable name. Three of the output files are automatically named `filename.output, filename.runin, filename.errors`. The name of the fourth output file is chosen by the user in `filename.dat`. The files `filename.runin, filename.errors` record errors in reading and processing the input file and can be used for trouble-shooting. The file `filename.output`, also referred to as the "print file", can be

used to look at specific data points, and holds any generated linear models. The fourth file contains the data suitable for plotting in Matlab. It is often referred to as the "plot file".

Although the input files are thoroughly discussed in [1], a quick tutorial is provided on the most relevant aspects to this work in terms of an example contained in section A.3.

The input file is organized in seven blocks of similar function type. The blocks are named SCHEDULES, INPUTS, RESTART, OUTPUT, INTEGRATION, EXCEPTIONS, and BALANCE EXCEPTIONS. Each block is started with a DEFINE command and ended with an END command.

```
DEFINE (block type)
 ...
END (block type)
```

The blocks are read into the simulation in a top-down fashion, so the user must sequence the blocks accordingly.

A.A.2.1 SCHEDULES

The SCHEDULES block is used to define univariant or bivariant curves representing functional relationships for the model. The curves can be functions of time or steady state point. The example file defines a schedule called `SCHFGR`. It is a schedule of thrust (XFCFGR) with time. The schedule data is listed in standard map reader format; the first two entries are zero, the third and forth indicate the number of data points of the first and second independent parameters. The remaining entries are data of the first independent parameter, the second independent parameter, and the dependent parameter respectively. The example file's schedule is univariant with time. It has a total of eight data points. Hence the third entry is "8.0" and the forth is "0.0".

The simulation assumes ramps between each defined point, so a step input can be approximated by placing points close together. In the example, a 200 lbf step is defined at 21.00 seconds. The input power code schedule, along with the output OPR, is plotted in Figure A.1.

A.A.2.2 INPUTS

The INPUTS block is used to define all inputs to the simulation. The example defines the altitude (ft), mach number, side slip(degrees), and angle of attack (degrees) to be zero. It

defines the engine face static pressure to be 14.696 psi. The thrust is referenced to the schedule defined in the preceding SCHEDULES block as SCHFGR.

A.A.2.3 RESTART

The RESTART block is used to specify a value for the GUESS routine in the simulation. As in the example file, "0.0" is normally used.

A.A.2.4 1st OUTPUT

The OUTPUT block defines the parameters the simulation is to record in its output files. The example file tells the simulation to record all the data points of OPR, EPR, N2, OPRREQ, EPRREQ, N2REQ, and XFCFGR. It instructs the simulation to print them to a file called `test.data`.

A.A.2.5 1st RUN

The RUN block provides the simulation with general information on the type of run required. The first RUN block in the example file tells the simulation to run one point to steady state using a maximum of of 200 iterations.

A.A.2.6 BALANCE EXCEPTIONS

The BALANCE EXCEPTIONS block is used to turn on or off simulation balances.

A.A.2.7 2nd OUTPUT

The 2nd OUTPUT block in the example file turns off printing to the print file (the `.output` file in our case).

A.A.2.8 2nd RUN

The 2nd RUN block in the example file runs a transient simulation and writes the data requested in the OUTPUT block every 0.05 seconds. It stops the simulation at 30 seconds,

no matter what is provided in the SCHEDULE block or elsewhere in the input file.

A.A.2.9 Hints

- There is a limit as to how many data points can be plotted in a run. If the simulation runs, but the plot file has only recorded a single data point, although you instructed otherwise, try increasing the plot time so you're not asking for as many points.
- While writing input files. Do not use tabs. Only use spaces.
- A * in the first column indicates a comment line

A.3 Example input file

```
*<DEBUG>
*************************************************************************
*                                                                       *
*      SCIP TRANSIENT ENGINE MODEL (CCD1453-00.0)                       *
*                                                                       *
*                     EXAMPLE FILE                                      *
*                                                                       *
*                                                                       *
*************************************************************************
 DEFINE SCHEDULES
*
* THRUST REQUEST TO ENGINE CONTROL AS A FUNCTION OF TIME
*
   SCHEDULE: SCHFGR  IS  XFCFGR    =   F(TIME ) ;
   SET SCHEDULE: SCHFGR=    0.0,     0.0,    8.0,   0.0,
       0.00,    5.00,    9.5,    12.5,     19.00,  21.00,
        21.01,  30.0,
      27000.0, 27000.0, 27100.0, 27100.0, 26900.0, 26900.0,
        27100.0, 27100.0;
*
 END    SCHEDULES
*************************************************************************
* INPUTS                                                                *
```

```
*************************************************************************
*
  DEFINE INPUT
*

* FLIGHT CONDITION
     ALT       = 0.0,
     XM        = 0.0     ,
*
* FLIGHT CONTROL INTERFACE
*
     XFCFGR    = SCHEDULE SCHFGR    ,
     XFCAOA    = 0.0     ,
     XFCBET    = 0.0     ,
     XFCPO     = 14.696  ,
     XFCMNL    = 0.0     ,
*
     IENGCON   = 1       ;
*
  END INPUT
*******************************
*  CALL FIRST GUESS ROUTINE   *
*******************************
  DEFINE RESTART
    GUESS = 0.0;
  END RESTART
*****************************
* OUTPUT OPTIONS           *
*****************************
  DEFINE OUTPUT
     STEADY-STATE PRINT  :       OFF          ;
     TRANSIENT PRINT     :       OFF          ;
     PRINT               :       ALL,         ;
     PLOT                :       ALL,
                                  OPR,    EPR,    N2,
                                  Y1MVC, Y2MVC, Y3MVC,
                                  OPRREQ, EPRREQ, N2REQ,
                                  XFCFGR;
```

```
     PLOT FILE            :   test.data  ;
     PLOT TITLE           :   EXAMPLE FILE DATE;
  END OUTPUT
*************************************************************************
*                                                                       *
*************************************************************************
   DEFINE RUN
    STEADY STATE : POINTS = 1., MAXPASS = 200;
   END RUN
*********************************************************
*  TURN OFF STEADY STATE ENGINE CONTROL BALANCES      *
*********************************************************
  DEFINE BALANCE EXCEPTIONS
     ACTIVATION FOR ANOZLBL2          :  OFF      ;
     ACTIVATION FOR WFPRBBL2          :  OFF      ;
     ACTIVATION FOR WFAB1BL2          :  OFF      ;
     ACTIVATION FOR WFAB2BL2          :  OFF      ;
     ACTIVATION FOR WFAB3BL2          :  OFF      ;
     ACTIVATION FOR VNHPCBL2          :  OFF      ;
     ACTIVATION FOR VNFANBL2          :  OFF      ;
     ACTIVATION FOR VNECSBL2          :  OFF      ;
     ACTIVATION FOR QDOTEHEX          :  OFF      ;
  END BALANCE EXCEPTIONS
*********************************************************
*  TURN OFF PRINT OUTPUT (PLOT OUTPUT IS STILL ON)    *
*********************************************************
   DEFINE OUTPUT
      PRINT  : NOPRINT ;
   END OUTPUT
***********************
*  RUN THE TRANSIENT  *
***********************
   DEFINE RUN
      TRANSIENT: DT = .0125 , PRINT TIME=5.0,  PLOT TIME=0.05,
      STOP TIME= 30.00 ;
   END RUN
```

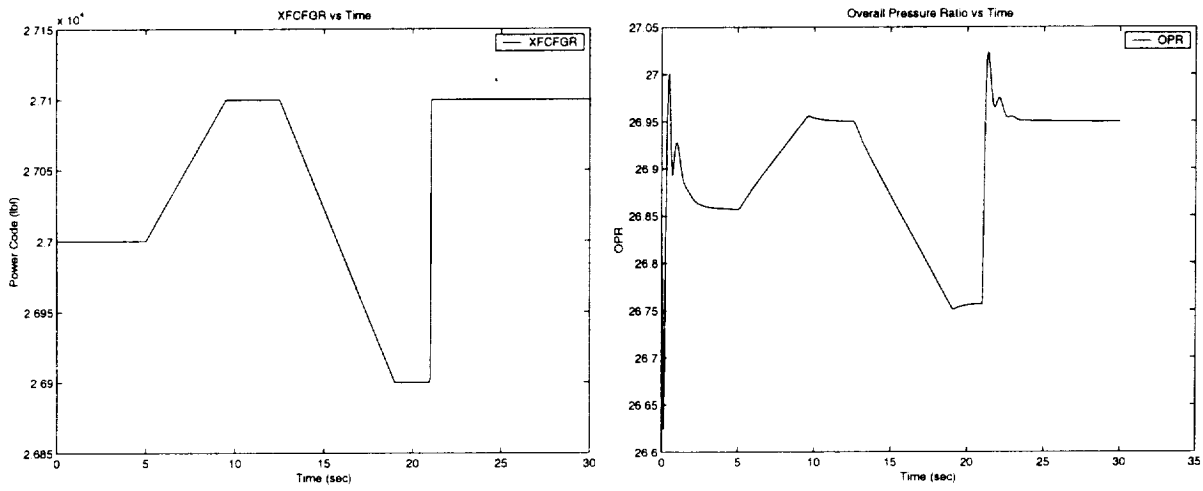

Figure A.1: Power Code and Plant output ($P4/P2$) from Example File

A.4 UNIX script files and Matlab M files

A.A.4.1 runorig

The script file `runorig` is used to run the original simulation. After creating an input file as explained in Section A.2 type

```
max% runporig filename
```

This will run the simulation, creating the four output files.

A.A.4.2 runptdes

The script file `runptdes` can be used to run a single point controller. Suppose a point design controller in a μ tool system matrix format named `sys` is to be tested in the nonlinear

simulation. The first step is to convert the controller into a Kblock format. This can be done using the matlab function sys2Kblock.m. The resulting matrix should then be saved in ascii format to a file named `Kblock`.

```
>> Kblock=sys2Kblock(sys,f21);
>> save -ascii Kblock Kblock ;
```

The next step is to create an input file as described in Section A.2. If the input file is named `filename.dat`, the nonlinear simulation can be run by typing

```
max% runptdes filename
```

This will run `filename.dat` using the controller contained in the file Kblock. Note that `runptdes` only runs on sun4 architecture, and Kblock must be in the same directory as test.dat.

A.A.4.3 runlpv

To run a LPV controller scheduled on power code in the nonlinear simulation, the script file `runlpv` can be used. The LPV controller must be in a ascii column format as detailed in the paper, and saved as `LPVdata`.

```
>> save -ascii LPVdata lpv_controller;
```

An input file, as outlined in Section A.2, can then be run with `runlpv`. For example, if the input file is named `filename.dat`

```
max% runlpv filename
```

Note that `LPVdata` must be in the same directory as test.dat.

A.A.4.4 data2m.m

Once a nonlinear simulation has been run, the data in the plot file can be easily loaded in matlab with data2m.m. The syntax is

```
[m,names]=data2m(datafile,flag);
```

Here `datafile` is a string containing the name of the data file. `Flag` is either "time" or "point". If your simulation was a time response and saved in `filename.data`,use the command

```
>>[m,names]=data2m('filename.data','time');
```

If your simulation was scheduled as something other than time and saved in `filename.data`, use the command

```
>>[m,names]=data2m('filename.data','point');
```

The matrix `names` contains the names of all variable contained in the varying matrix `m`.

A.A.4.5 engplot.m

Once data from a data file had been loaded into Matlab, it can be plotted with `engplot`. For example if the data matrix `m` contains data for `OPR`, you can use the command

```
>>engplot(m,names,'OPR');
```

A.5 Plant Linearization

Linear plants can be created by the nonlinear simulation with a `LINEARIZATION` block in the input file. As before, I will illustrate usage with an example file. The file can be found in section A.6

The blocks used for linearization follow the same format as those described above. Two blocks, `LINEARIZATION` and `LINEARIZATION EXCEPTIONS` define the linearization to be performed. Typically the base point about which linearization is desired, is defined through a steady state run

```
DEFINE RUN
  STEADY STATE : POINTS = 1., MAXPASS = 200;
END RUN
```

A.A.5.1 INTEGRATION DEFAULTS and EXCEPTIONS

The states are controlled through the INTEGRATION DEFAULTS and EXCEPTIONS blocks and the balances through the BALANCE DEFAULTS and EXCEPTIONS blocks.

```
DEFINE INTEGRATION EXCEPTIONS
  ACTIVATION FOR SN2      : STEADY STATE ;
  ACTIVATION FOR PTECHD   : STEADY STATE ;
  ACTIVATION FOR TTECHD   : STEADY STATE ;
  ACTIVATION FOR SNECS    : STEADY STATE ;
  ACTIVATION FOR PT65     : STEADY STATE ;
END INTEGRATION EXCEPTIONS
DEFINE BALANCE EXCEPTIONS
  ACTIVATION FOR ANOZLBL2          :  OFF     ;
  ACTIVATION FOR WFPRBBL2          :  OFF     ;
  ACTIVATION FOR WFAB1BL2          :  OFF     ;
  ACTIVATION FOR WFAB2BL2          :  OFF     ;
  ACTIVATION FOR WFAB3BL2          :  OFF     ;
  ACTIVATION FOR VNHPCBL2          :  OFF     ;
  ACTIVATION FOR VNFANBL2          :  OFF     ;
  ACTIVATION FOR VNECSBL2          :  OFF     ;
  ACTIVATION FOR QDOTEHEX          :  OFF     ;
END BALANCE EXCEPTIONS
```

A.A.5.2 LINEARIZATION PRINT

The LINEARIZATION PRINT option must be turned on in the OUTPUT block

```
DEFINE OUTPUT
  LINEARIZATION PRINT : ON    ;
END OUTPUT
```

A.A.5.3 LINEARIZATION

The LINEARIZATION block is used to define the inputs and outputs of the desired linear model.

```
DEFINE LINEARIZATION
  INPUTS  : WFPRIB   , AREANOZL,  VANEHPC;
  OUTPUTS :   PT2,    PT4,   PT6,   SN2;
END LINEARIZATION
```

A.A.5.4 Output file

The resulting A,B, C, and D matrices are written to the output file. These matrices can be automatically loaded into matlab using `output2sys.m` (see section A.A.7.1). Note the nonlinear simulation may return error messages in the output file. These errors can generally be ignored, because of a known error checking problems with the program.

A.6 Example linearization input file

```
DEFINE INPUT
* FLIGHT CONDITION
     ALT       = 0.0     ,
     XM        = 0.0     ,
* FLIGHT CONTROL INTERFACE
     XFCFGR    =  3000   ,
     XFCAOA    = 0.0     ,
     XFCBET    = 0.0     ,
     XFCPO     = 14.696  ,
     XFCMNL    = 0.0     ,
     IENGCON   = 1       ,
END INPUT

DEFINE RESTART
  GUESS = 0.0;
END RESTART

DEFINE OUTPUT
  PRINT                :      DUMPALL      ;
  STEADY-STATE PRINT   :       OFF         ;
  TRANSIENT PRINT      :       OFF         ;
  PLOT                 :       ALL,
```

```
                              WFPRIB, AREANOZL, VANEHPC,
      PT2, PT4, PT6, SN2;
  PLOT FILE            :      tmp.data     ;
  PLOT TITLE           :      LINEARIZE  ;
END OUTPUT

DEFINE RUN
  STEADY STATE : POINTS = 1., MAXPASS = 200;
END RUN

 DEFINE INTEGRATION EXCEPTIONS
    ACTIVATION FOR SN2       : STEADY STATE ;
    ACTIVATION FOR PTECHD    : STEADY STATE ;
    ACTIVATION FOR TTECHD    : STEADY STATE ;
    ACTIVATION FOR SNECS     : STEADY STATE ;
    ACTIVATION FOR PT65      : STEADY STATE ;
 END INTEGRATION EXCEPTIONS


* TURN OFF STEADY STATE ENGINE CONTROL BALANCES      *
DEFINE BALANCE EXCEPTIONS
  ACTIVATION FOR ANOZLBL2          :  OFF      ;
  ACTIVATION FOR WFPRBBL2          :  OFF      ;
  ACTIVATION FOR WFAB1BL2          :  OFF      ;
  ACTIVATION FOR WFAB2BL2          :  OFF      ;
  ACTIVATION FOR WFAB3BL2          :  OFF      ;
  ACTIVATION FOR VNHPCBL2          :  OFF      ;
  ACTIVATION FOR VNFANBL2          :  OFF      ;
  ACTIVATION FOR VNECSBL2          :  OFF      ;
  ACTIVATION FOR QDOTEHEX          :  OFF      ;
END BALANCE EXCEPTIONS

DEFINE OUTPUT
  LINEARIZATION PRINT : ON    ;
END OUTPUT

DEFINE LINEARIZATION
  INPUTS  : WFPRIB    , AREANOZL,  VANEHPC;
```

```
  OUTPUTS :   PT2,    PT4,   PT6,   SN2;
END LINEARIZATION

DEFINE LINEARIZATION EXCEPTIONS
  NORMALIZER   FOR  VANEHPC  : 100.0 ;
END LINEARIZATION EXCEPTIONS

DEFINE BALANCE EXCEPTIONS
  ACTIVATION FOR ANOZLBL2  : OFF   ;
  ACTIVATION FOR WFPRBBL2  : OFF   ;
  ACTIVATION FOR VNHPCBL2  : OFF   ;
  ACTIVATION FOR VNFANBL2  : OFF   ;
  ACTIVATION FOR VNECSBL2  : OFF   ;
  ACTIVATION FOR WFAB1BL2  : OFF   ;
  ACTIVATION FOR WFAB2BL2  : OFF   ;
  ACTIVATION FOR WFAB3BL2  : OFF   ;
  ACTIVATION FOR QDOTEHEX  : ON    ;
END BALANCE EXCEPTIONS

DEFINE RUN
  LINEARIZE : ;
END    RUN
```

A.7 MORE UNIX SCRIPT and MATLAB files

A.A.7.1 output2sys.m

The Matlab M file output2sys.m reads the linearized plant from the simulation output file to generate a system matrix. If the output file is named `filename.output`, for example, the following command will create a system matrix names `sys`

```
>> sys=output2sys('filename');
```

A.A.7.2 linearize

The UNIX script file **linearize** can be used to generate the plant models at 3000, 6000, 9000, ... 30000 lbf thrust by replacing the line

```
XFCFGR   =  3000   ,
```

in the input file with

```
XFCFGR   =  <INSERT_POINT>
```

The script replaces <INSERT_POINT> with the values listed in the ascii file **points** generating a series of linearized plant models.

For example, to create a series of linearized plants at sea level, the INPUT block should have the form

```
DEFINE INPUT
* FLIGHT CONDITION
     ALT       = 0.0     ,
     XM        = 0.0     ,
* FLIGHT CONTROL INTERFACE
     XFCFGR    =  <INSERT_POINT>
     XFCAOA    = 0.0     ,
     XFCBET    = 0.0     ,
     XFCPO     = 14.696  ,
     XFCMNL    = 0.0     ,
     IENGCON   = 1       ,
*     IENGCON  = 0       ,
END INPUT
```

The file **points** is the simple ascii file:

```
 3000.0
 6000.0
 9000.0
12000.0
15000.0
18000.0
```

21000.0
24000.0
27000.0
30000.0

Appendix B

FORTRAN Source Code

This appendix contains the FORTRAN subroutines written for this project. Small explanations are included before each.

B.1 Point Design Implementation

This section contains the FORTRAN code written to implement point design controllers into the nonlinear simulation.

B.B.1.1 compute_state

The subroutine `compute_state` is called by the larger simulation. It, in turn, calls all the other subroutines which read in the controllers and updates the controller states and outputs.

```
      Subroutine compute_state(FGRREQ,CRCYB1, CRCYB2, CRCYB3,U1in,
     &                         U2in, U3in, Y1out, Y2out, Y3out)

C     FORMAL PARAMETERS
      REAL U1in, U2in, U3in

      REAL Y1out, Y2out, Y3out
      REAL Y1outa, Y2outa, Y3outa
      REAL FGRREQ
```

```
      REAL Min(24,1), Mout(24,1)
      REAL P(72,8)
      REAL A(24,24)
      REAL St(21)

      REAL Y1base, Y2base, Y3base
      Integer counter

      save  St, counter, A
      data (St(i), i=1,21)/21*0.0/
      data counter /0/
      counter=counter+1

      if(counter.eq.1) then
           CALL GETMXA8(P,72)
           Call REFORMMX(P,72,8,A,24,24)
      endif

C---MULTIPLY ACROSS THE CONTROLLER ------------------------------------

       Do 11 i=1,21
           Min(i,1)=St(i)
 11    Continue
      Min(22,1)=U1in
      Min(23,1)=U2in
      Min(24,1)=U3in

      CALL MULTMX(A,24,24,Min,24,1,Mout,24,1)

      Do 22 i=1,21
         St(i)  = Mout(i,1)
 22    Continue
      Y1outa   = Mout(22,1)
      Y2outa   = Mout(23,1)
      Y3outa   = Mout(24,1)

      CALL linterp(CRCYB1,FGRREQ,Y1base)
```

```
      CALL linterp(CRCYB2,FGRREQ,Y2base)
      CALL linterp(CRCYB3,FGRREQ,Y3base)

      Y1out = Y1outa + Y1base
      Y2out = Y2outa + Y2base
      Y3out = Y3outa + Y3base

      RETURN
      END
```

B.B.1.2 linterp

The subroutine **linterp** is used to linearly interpolate between two points. The vector T contains x-y data in standard mapping format. A value of x is input as Xin and linterp returns the corresponding value of y in Yout.

```
      Subroutine linterp(T,Xin,Yout)

C PURPOSE
C Linearly Interpolate between points on a function f(x).
C Given point on f(x), linterp returns f(x) for any value of x.
C
C
C SYNTAX
C  CALL linterp(T,Xin,Yout)
C
C   T : Data in PW's mapping format (which could be called with tabx2)
C

C Variables passed in:
      DIMENSION  T(*)
      REAL Xin, Yout
C Internal Variables:

      REAL nx,n

      REAL X(10), Y(10)
```

```
      INTEGER i
      REAL X_high, X_low, Y_high, Y_low
      nx = T(3)

      Do 500, n=1,nx
         X(n)=T(4+n)
         Y(n)=T(4+n+nx)
 500  CONTINUE

      if (Xin.lt.X(1)) then
          Yout=Y(2)-(Y(2)-Y(1))/(X(2)-X(1))*(X(2)-Xin)

      elseif (Xin.gt.X(nx)) then
          Yout=Y(nx-1)+(Y(nx)-Y(nx-1))/(X(nx)-X(nx-1))*(Xin-X(nx-1))

      else
          i=1
 600      if (Xin.gt.X(i)) then
             i=i+1
             go to 600
          endif

          X_high=X(i)
          X_low =X(i-1)

          Y_high=Y(i)
          Y_low =Y(i-1)

C         % 2nd: Linerly interpolate the exact value
C         %   the ratio xfcfgr/(pchigh-pclow) = opr/(oprhihg-oprlow)

          Yout=(Xin-X_low)/(X_high-X_low)*(Y_high-Y_low) + Y_low


      endif

      END
```

B.B.1.3 GETMXA8

The subroutine `GETMXA8` reads in the controller from a file named `Kblock`. It inputs the number of rows to read (r). It returns the matrix read as a $r \times 8$ column matrix A.

```
      SUBROUTINE GETMXA8(A,r)

      REAL c1, c2, c3, c4, c5, c6, c7, c8
      Integer r
      REAL A(r,8)
      INTEGER n

      n=0
      OPEN (Unit=15, File='Kblock', STATUS='OLD')
123   READ (15,124,END=125) c1, c2, c3, c4, c5, c6, c7, c8
124   FORMAT(2X,E14.7,2X,E14.7,2X,E14.7,2X,E14.7,
     &       2X,E14.7,2X,E14.7,2X,E14.7,2X,E14.7)
      n=n+1
      A(n,1) = c1
      A(n,2) = c2
      A(n,3) = c3
      A(n,4) = c4
      A(n,5) = c5
      A(n,6) = c6
      A(n,7) = c7
      A(n,8) = c8
      GOTO 123
125   n=0
      CLOSE(UNIT=15)
      RETURN
      END
```

B.B.1.4 REFORMMX

The subroutine `REFORMMX` reforms the matrix A (read in by GETMX8) into the correct three input, three output, 21 state format.

```
      SUBROUTINE REFORMMX(A,R1,C1,B,R2,C2)
C THIS SUBROUTINE REFORMS THE 72 by 8 matrix into a 24 by 24
      INTEGER R1, R2, C1, C2, BL, R, C, BLOCKS, AR
      REAL A(R1,C1)
      REAL B(R2,C2)

      BLOCKS=R1/R2

         Do 2 BL=1,BLOCKS
           Do 3 R=1,R2
               Do 4 C=1,C1
                    BC=C+C1*(BL-1)
                    AR=R+R2*(BL-1)
                    B(R,BC)=A(AR ,C)
 4             Continue
 3          Continue
 2       Continue

         RETURN
         END
```

B.B.1.5 MULTMX

The subroutine MULTMX inputs the matrices A and B along with their dimensions, and returns the product of $A \cdot B$ in the matrix C.

```
      SUBROUTINE MULTMX(A,AROW,ACOL,B,BROW,BCOL,
     & C,CROW,CCOL)
C
C     THIS SUBROUTINE MULTIPLIES ARRAYS A AND B AND STORES RESULT AS C
C
      INTEGER AROW,ACOL,BROW,BCOL,CROW,CCOL,I,J,K
      REAL A(AROW,ACOL),B(BROW,BCOL),C(CROW,CCOL)
      LOGICAL ERROR

      ERROR= .FALSE.
      IF (ACOL.NE.BROW) ERROR = .TRUE.
```

```
      IF (AROW.NE.CROW) ERROR = .TRUE.
      IF (BCOL.NE.CCOL) ERROR = .TRUE.

      IF (ERROR) THEN
        PRINT*, 'ERROR IN THE ARRAY SIZES'
        Stop
      ELSE
        DO 30 I=1,CROW
         DO 25 J=1,CCOL
         C(I,J) = 0.0
            DO 15 K=1,ACOL
            C(I,J) = C(I,J) + A(I,K)*B(K,J)
15          CONTINUE
25       CONTINUE
30     CONTINUE
      ENDIF
      RETURN
      END
```

B.2 Power Code LPV Implementation

B.B.2.1 compute_state

The LPV Implementation, as with the point design code, uses compute_state as it's main routine and calls all other subroutines from it.

```
      Subroutine compute_state(FGRREQ, CRCYB1, CRCYB2, CRCYB3, U1in,
     &                         U2in, U3in, Y1out, Y2out, Y3out)



C     FORMAL PARAMETERS
      REAL U1in, U2in, U3in
      REAL OREQ, EREQ, NREQ
      REAL Y1out, Y2out, Y3out
      REAL Y1outa, Y2outa, Y3outa
```

```
      REAL FGRREQ

      REAL Min(33,1), Mout(33,1)
      REAL A(33,33)
      REAL St(30)
C YBASE VARIABLES
      REAL Y1base, Y2base, Y3base

      save  St
      data (St(i), i=1,30)/30*0.0/


      CALL GETMXA(FGRREQ,A)


C---MULTIPLY ACROSS THE CONTROLLER ------------------------------------

      Do 11 i=1,30
           Min(i,1)=St(i)
 11   Continue
      Min(31,1)=U1in
      Min(32,1)=U2in
      Min(33,1)=U3in

      CALL MULTMX(A,33,33,Min,33,1,Mout,33,1)


      Do 22 i=1,30
         St(i)  = Mout(i,1)
 22   Continue
      Y1outa   = Mout(31,1)
      Y2outa   = Mout(32,1)
      Y3outa   = Mout(33,1)

      CALL linterp(CRCYB1,FGRREQ,Y1base)
      CALL linterp(CRCYB2,FGRREQ,Y2base)
      CALL linterp(CRCYB3,FGRREQ,Y3base)
```

```
      Y1out = Y1outa + Y1base
      Y2out = Y2outa + Y2base
      Y3out = Y3outa + Y3base

      RETURN
      END
```

B.B.2.2 GETMXA

The subroutine **GETMXA** accepts a powercode as input and returns the linearly interpolated controller for that powercode.

```
C=======================================================================
      SUBROUTINE GETMXA(PC,BIGA)
************************************************************************
* THIS IS THE LPV SUBROUTINE
* This subroutine reads in a location within a given power code matrix to
*find percent values of the four corners around it.  Then new A,B,C and D
*matricies are computed from these percent values.  If the location
*given does not fall within the power code matrix the subroutine will
*not run and will return a value of ingrid=0.
*
* Required input to subroutine:
*
*       G - Gigantic matrix, power code and altitude matrix organized
*               left to right, top to bottom such that all power codes
*               at a given altitude run consecutively.  Also single column
*               so single matircies A through D also run across the rows.
*               ex      [G] =   [[A]        WHERE       [A] =   [(1,1)
*                               [B]                             (1,2)
*                               [C]                             (1,3)
*                               [D](1,1)                          .
*                               [A]                               .
*                               [B]                             (2,1)
*                               [C]                             (2,2)
*                               [D](1,2)                          .
```

```
*                                   .                                   .
*                                   .                                   . ]
*                                 [A]
*                                 [B](2,1)
*                                  .
*                                  . ]
*
*       PC - POWER CODE
*       VEC - power code and altitude matrix (WAT2)
*       il - length of PC vector (3,6,9,12,15,18,21,24,27,30) so 10
*
* Output of subroutine:
*
*       ingrid - returns 1 if located within WAT2 matrix and
*                returns 0 if not located within WAT2 matrix and stops
*       A,B,C,D - new matricies computed from G
*
* Temporary variables:
*
*       S - temporary matrix not passed through
*       alpha - percent location of WAC2 between points
**********************************************************************
c
    integer nr,nc,nx,ny,nu,il,i,j,tl,bl,ii,jj,nt
    parameter(il=10,nx=30,ny=3,nu=3)
    real A(30,30),B(30,3),C(3,30),D(3,3)
    real alpha,Stmp(1089),VEC(10),BIGA(33,33)
    real PC, temp
    real G(10890)
    Integer counter

C    ReadLPV VARIABLES
    Integer nxtrue
    Real LPVin(10890)

    save counter, G
    data counter /0/
    counter=counter+1
```

```
      nc=nx+nu
      nr=nx+ny
*
* Get PC vector
*
      VEC(1)=  3000.0
      VEC(2)=  6000.0
      VEC(3)=  9000.0
      VEC(4)= 12000.0
      VEC(5)= 15000.0
      VEC(6)= 18000.0
      VEC(7)= 21000.0
      VEC(8)= 24000.0
      VEC(9)= 27000.0
      VEC(10)=30000.0
*
* Logic function to make temp in bounds
*
      temp=PC
      if(PC.LT.VEC(1))temp=VEC(1)
      if(PC.GT.VEC(il))temp=VEC(il)
*
* READ in G (found through ReadLPV and PadLPV)
*
      if(counter.eq.1) then
           Call readlpv(nxtrue,LPVin)
           Call padLPV(nxtrue,LPVin,G)
      endif
*
* Find contributions of each side and calculate start and end
*  point of first matrix (tl=top limet, bl= botom limit)
*
      alpha=0.0
      do 20 i=1,il-1
       if(temp.GE.VEC(i).AND.temp.LE.VEC(i+1))then
        if(temp.NE.VEC(i))then
         alpha=(temp-VEC(i))/(VEC(i+1)-VEC(i))
```

```
         end if
         tl=nr*nc*(i-1)+1
         bl=nr*nc*i
         goto 30
        end if
20     continue
*
* Compute new A,B,C, and D matricies
*
30     nt=nr*nc
       ii=0
       do 40 i=tl,bl
        ii=ii+1
c
        Stmp(ii)=G(i)+alpha*(G(i+nt)-G(i))
c
40     continue
c --------------Make A---------------------
       ii=0
       do 60 i=1,nx
        do 50 j=1,nx
         ii=ii+1
         A(i,j)=Stmp(ii)
50      continue
60     continue
c -------------Make B----------------------
       ii=nx*nx
       do 80 i=1,nx
        do 70 j=1,nu
         ii=ii+1
        B(i,j)=Stmp(ii)
70      continue
80     continue
c --------------Make C---------------------
       ii=nc*nx
       do 100 i=1,ny
        do 90 j=1,nx
         ii=ii+1
```

```
       C(i,j)=Stmp(ii)
90      continue
100    continue
c --------------Make D--------------------
       ii=nc*nx+nx*ny
       do 120 i=1,ny
        do 110 j=1,nu
         ii=ii+1
         D(i,j)=Stmp(ii)
110     continue
120    continue
*
* -----------Make it one large matrix -----
*
       do 123 i=1,nx
       jj=nx
        do 121 j=1,nx
        BIGA(i,j)=A(i,j)
121    continue
        do 122 j=1,nu
        jj=jj+1
        BIGA(i,jj)=B(i,j)
122     continue
123    continue
c
       ii=nx
       do 126 i=1,ny
       jj=nx
       ii=ii+1
        do 124 j=1,nx
         BIGA(ii,j)=C(i,j)
124     continue
        do 125 j=1,nu
         jj=jj+1
        BIGA(ii,jj)=D(i,j)
125     continue
126    continue
```

```
      end
```

B.B.2.3 padLPV

The subroutine **padLPV** inputs the LPV vector read in by **readlpv** along with the number of states of the LPV controller, and returns a LPV vector that has been padded with zeros so that all controllers have 30 states.

```
C-------------------------------------------------
C This subroutine rearranges LPVin by moving the zeros
C according to nx.   (Largest possible is 30 states, 10 pc vec)
C-------------------------------------------------
      Subroutine padLPV(nx,LPVin,LPVout)


      Integer nx, i, j, loc, locin, pclength
      Real LPVin(10890),  LPVout(10890)


      pclength=10
      loc=0
      locin=0

      Do 99 k=1,pclength

C----A-----------------------------------
         Do 1 i=1,30*(30-nx)
            loc=loc+1
            LPVout(loc)=0
 1       Continue

         Do 2 i=1,nx
            Do 21 j=1,30-nx
               loc=loc+1
               LPVout(loc)=0
 21         Continue
            Do 22 j=1,nx
```

```
                loc=loc+1
                locin=locin+1
                LPVout(loc)=LPVin(locin)
 22          Continue
 2        Continue

C----B-----------------------------------

          Do 3 i=1,3*(30-nx)
             loc=loc+1
             LPVout(loc)=0
 3        Continue
          Do 4 i=1,nx*3
             loc=loc+1
             locin=locin+1
             LPVout(loc)=LPVin(locin)
 4        Continue

C----C-----------------------------------

          Do 5 i=1,3
             Do 51 j=1,30-nx
                loc=loc+1
                LPVout(loc)=0
 51          Continue
             Do 52 j=1,nx
                loc=loc+1
                locin=locin+1
                LPVout(loc)=LPVin(locin)
 52          Continue
 5        Continue

C----D-----------------------------------

          Do 6 i=1,9
             loc=loc+1
             locin=locin+1
             LPVout(loc)=LPVin(locin)
```

```
 6       Continue

 99    Continue

       End
C-----------------END subroutine-------------------------------
```

B.B.2.4 readLPV

The subroutine `readlpv` reads in the LPV vecctor from an external file named `LPVdata` and returns it along with the number of states contained in each of the vectors controllers. It assumes there are a total of 10 controllers in the LPV vector.

```
C-------------------------------------------------------------
C This subroutine reads in the LPV controller from LPVdata
C It must be called before compute_state b/c it provides the
C number of states to be used in compute_state
C (Assumes PCvec length of 10, Max number of states=30)
C-------------------------------------------------------------
      Subroutine readlpv(nxi,LPV)

      Integer num, nxi, counter
      Real LPV(10890), LPVlocal(10890), nx
      save counter, LPVlocal, nx

      data counter /0/
      counter=counter+1

      if(counter.eq.1) then
         num=0
 9       OPEN (Unit=19, File='LPVdata', STATUS='OLD')
            num=num+1
            READ(19,10,end=11) LPVlocal(num)
 10         FORMAT(2X,F14.7)
         go to 9

 11      CLOSE(UNIT=19)
```

```
      endif

      nx=sqrt(real(num/10))-3
      nxi=int(nx)

      do 12 i=1,num
         LPV(i)=LPVlocal(i)
 12   continue

      END
C-----------------END subroutine-----------------------------
```

The LPV implementation also calls the subroutines `linterp` and `mutlmx` which are listed in the Point Design section above.

B.3 Compilation

The full engine simulation consists of three FORTRAN files and an assortment of auxiliary files containing common data blocks. The main files are

```
ccd1453_eng_con.f
ccd1453_main.f
ccd1453_roc_util.f
ccd1453_eng_mod.f
ccd1453_roc_run.f
```

The auxiliary files need are

```
CLCASCOM
CLCCOM
ECSCASCOM
ECSCCOM
NCCCOM
NCCOM
NCSCOM
SHRCCOM
```

To compile the program with a new control algorithm copy in into `ccd1453_eng_con.f`. For example, if the new control algorithm was in a file named `newcontrol.f`, type

```
cp newcontrol.f ccd1453_eng_con.f
```

Then run the makefile on a `sun4` architecture by typing

```
%max make
```

This runs the UNIX script file `Makefile.sun4`. It was written to compile, and link all the components.

Appendix C

Line by Line Example

This Appendix lists the specific commands to run a ROCETS simulation using

- The PW simulation
- A point design simulation
- A LPV simulation

Note that this Appendix is strictly intended for those working in Dr. Gary Balas' lab who have access to the Sun machines `max` and/or `marlowe`. At this point, the ROCETS simulations only run on sun4 architecture. The example files are located at `/home/res6/jackryan/pwfl/Examp` These directions are also located at `/home/res6/jackryan/pwfl/Examples/HowToRun.txt`.

Before running the following examples, type the line

```
%max set path=(/home/res6/jackryan/Script /home/res6/jackryan/pwfl/PwScripts $path)
```

It adds the path which contains all the necessary UNIX script files. Also add the paths

```
>>   path('/home/res6/jackryan/pwfl/m',path);
>>   path('/home/res6/jackryan/pwfl/m/MakeModel',path);
>>   path('/home/res6/jackryan/pwfl/m/TestModel',path);
```

in Matlab. They contain all the necessay Matlab M files.

C.1 The original Pratt & Whitney simulation

To run the original Pratt & Whitney simulation, go to the directory `/home/res6/jackryan/pwfl/Exampl` and type

```
max% pwd
/home/res6/jackryan/pwfl/Examples
max% runorig  example
Running example (sun4 version) ..
max%
```

It creates the files

```
example.errors     : Contains any error messages if any occurred
example.runin      : a filtered copy of example.dat which is
                     sent to the program.  Lists .dat file errors
                      the end (good for trouble shooting)
example.output     : values of a single point in the run
fort.25            : a link to example.dat
test.data          : The plot file.  Lists all the data of the run.
```

Now load and plot the data into Matlab by typing the commands at a Matlab prompt

```
>> [m,names]=data2m('test.data','time');
First line of data: 10
Number of data points: 722

Reading variables:
TIME   EPR    EPRREQ FGR    N2     N2REQ
OPR    OPRREQ WAC2   XFCFGR XFCFGR Y1MVC
Y2MVC  Y3MVC

Number of variables found: 14
>> engplot(m,names,'EPR','EPRREQ');
The variable EPR is is ambiguous.
Using the first occurrence of EPR
>> print -deps epr_plot1
>>
```

C.2 Running a Point Design

Now run the Point Design version. The file `Kblock` contains a point designed controller. The controller *must* be in a file named `Kblock`. First move `test.data` to `test.orig.data` so that the PW simulation results are not over-written.

```
max% mv test.data test.lpv.data
max% runptdes example
Running example (sun4 version) ..
max%
```

In Matlab, load and plot the new data

```
>> opr_plot(m,mp,names,'EPR');
The variable EPR is is ambiguous.
Using the first occurrence of EPR
The variable EPR is is ambiguous.
Using the first occurrence of EPR
>> print -deps epr_plot2
>>
```

C.3 Running a LPV Design

Now run the LPV version. `LPVdata` contains the LPV controller; the program looks for `LPVdata` so the controller *must* have this name. First move `test.data` to `test.orig.data` so the results from the point design simulation will not be over-written

```
max% mv test.data test.orig.data
max% runlpv example
Running example (sun4 version) ..
max%
```

In Matlab, load the new data and plot the new and old together

```
>> [ml,names]=data2m('test.data','time');
First line of data: 10
```

```
Number of data points: 722

Reading variables:
TIME   EPR    EPRREQ FGR    N2     N2REQ
OPR    OPRREQ WAC2   XFCFGR XFCFGR Y1MVC
Y2MVC  Y3MVC

Number of variables found: 14
>>  opr_plot(m,ml,names,'EPR');
The variable EPR is is ambiguous.
Using the first occurence of EPR
The variable EPR is is ambiguous.
Using the first occurrence of EPR
>> print -deps epr_plot3
>>
```

All files used and created in these examples are contained in `/home/res6/jackryan/pwfl/Examples/`.

There are a few more executables (all located in `/home/res6/jackryan/pwfl/PwScripts/`) which have not been specifically mentioned. They are run in the same fashion as listed above. They are

```
runorig    Runs the original ROCETS

runptdes   Run a point design controller

runlpv     Run an LPV controller scheduled on power code

runlpvl    Run an LPV controller scheduled on lagged power code

runtemp    Used to run temporary codes for testing (I just use it
             for ease ... it's not necessary)
```

All the FORTRAN source code used in this project is located at

`/home/res6/jackryan/pwfl/src/`

The various control algorithm source code files are listed and described below

lpv_lag.f	LPV code which schedules on lagged PC (run with runlpvl)
jack_ml_eng_con.lpv.f	LPV code scheduled on PC (run with runlpv)
jack_ml_eng_con.lpv2d.f	LPV code scheduled on 2 variables (In a proof of concept stage)
jack_ml_eng_con.ptdes.f	Point Design code (reads in Kblock) (run with runptdes)
wac2.lag.lpv.new.f	LPV code scheduled on filtered WAC2 (run with runwacl)
wac2.lpv.new.f	LPV code scheduled on WAC2 (run with runwac)